

```

*****
6236 Mon Feb 19 00:52:49 2018
new/usr/src/tools/cw/cw.1onbld
cw: don't shadow pure pre-processing
cw(1onbld): --shadow not --secondary
*****
1 .\"
2 .\" CDDL HEADER START
3 .\"
4 .\" The contents of this file are subject to the terms of the
5 .\" Common Development and Distribution License (the "License").
6 .\" You may not use this file except in compliance with the License.
7 .\"
8 .\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 .\" or http://www.opensolaris.org/os/licensing.
10 .\" See the License for the specific language governing permissions
11 .\" and limitations under the License.
12 .\"
13 .\" When distributing Covered Code, include this CDDL HEADER in each
14 .\" file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 .\" If applicable, add the following below this CDDL HEADER, with the
16 .\" fields enclosed by brackets "[]" replaced with your own identifying
17 .\" information: Portions Copyright [yyyy] [name of copyright owner]
18 .\"
19 .\" CDDL HEADER END
20 .\"
21 .\" Copyright 2010 Sun Microsystems, Inc. All rights reserved.
22 .\" Use is subject to license terms.
23 .\"
24 .Dd February 10, 2018
25 .Dt CW 1ONBLD
26 .Os
27 .Sh NAME
28 .Nm cw
29 .Nd invoke one or more compilers with argument translation
30 .Sh SYNOPSIS
31 .Nm cw
32 .Op Fl C
33 .Op Fl -versions
34 .Op Fl -noecho
35 .Fl -primary Ar compiler
36 .Op Fl -shadow Ar compiler ...
37 .Fl -
38 .Ar compiler args ...
39 .Sh DESCRIPTION
40 .Nm cw
41 is a facility for invoking one or more compilers, providing translation from
42 Sun style arguments as appropriate.
43 This allows the use of arbitrary compilers without the need to alter large
44 numbers of makefiles.
45 A mode called shadow compilation invokes multiple compilers so that warnings
46 and errors may be obtained from both.
47 See SHADOW COMPILATION for details.
48 This version of cw supports compilers with both Sun Studio 12 and GCC-style
49 command lines.
50 .Sh ARGUMENTS
51 Both the
52 .Fl -primary
53 and
54 .Fl -shadow
54 .Fl -secondary
55 parameters take a
56 .Em compiler specification .
57 This is a comma-separated list of the form
58 .Ar name,executable,style
59 Where

```

```

60 .Ar name
61 is a name for the compiler,
62 .Ar executable
63 is the full path to the compiler executable, and
64 .Ar style
65 is the style of command-line options the compiler expects, either
66 .Em sun
67 or
68 .Em gnu .
69 .Bl -tag -width indent
70 .It Fl -primary Ar compiler
71 Specify the compiler to be used primarily (that which is used for link-editing
72 and pre-processing, and whos objects we deliver).
73 .It Fl -shadow Ar compiler
74 Specify a shadow compiler, which builds sources for the sake of checking code
75 quality and compatibility, but has its output discarded.
76 .It Fl -noecho
77 Do not echo the actual command line of any compilers invoked.
78 .It Fl -versions
79 Request from each configured primary and shadow compiler its version
80 information.
81 .It Fl C
82 The sources being compiled are C++. This is necessary as it affects the
83 translation of compiler arguments.
84 .It Fl -
85 Arguments intended for the compilers themselves must be separated from those
86 of
87 .Nm cw
88 by a
89 .Fl - .
90 .It Fl _name=
91 .It Fl _style=
92 Parameters intended for the compiler be guarded with options of the form
93 .Fl _name=
94 and
95 .Fl _style=
96 Where
97 .Em name
98 and
99 .Em style
100 are those passed to
101 .Fl -primary
102 and
103 .Fl -shadow
104 this allows certain flags to be passed only to certain classes of compiler.
105 .Pp
106 For historical reasons, the
107 .Fl _style=
108 option is also translated such that a style of
109 .Em sun
110 may use the flag
111 .Fl _cc=
112 and a style of
113 .Em gnu
114 may use the flag
115 .Fl _gcc= ,
116 and when the
117 .Fl C
118 option is given and C++ is in use the style of
119 .Em sun
120 may use the flag
121 .Fl _CC=
122 and the style of
123 .Em gnu
124 may use the flag
125 .Fl _g++= .

```

```

126 .El
127 .Sh SHADOW COMPILATION
128 If
129 .Fl -shadow
130 compilers are specified
131 .Nm cw
132 will invoke shadow compiler, with the outputs modified (as well as any
133 translation for compiler style) as follows:
134 .Bl -enum
135 .It
136 If neither of
136 If none of
137 .Fl c ,
138 .Fl E ,
139 .Fl P ,
140 or
138 .Fl S
139 appears in the argument list (that is, linking is attempted or only the
140 pre-processor is invoked), the shadow compilers will not be invoked.
142 appears in the argument list (that is, linking is attempted), the shadow
143 compilers will not be invoked.
144 This is because the objects built with that compiler which would be linked
145 have been previously discarded.
141 .It
142 If the
143 .Fl o Ar filename
144 option was provided, with or without a separating space, it will be replaced wit
145 .Fl o Ar tempfile
146 .It
147 If the option
148 .Fl o
149 was not provided,
150 .Fl o Ar tempfile
151 will be added to the end of the argument list used to invoke
152 the shadow compilers.
153 .El
154 When shadow compilation is in effect,
155 .Nm cw
156 writes to standard error each compiler's standard error output following its
157 argument list.
158 Messages from the compilers will not be interleaved.
159 If
160 .Nm cw
161 is used to invoke the preprocessor and no output location is specified,
162 .Nm cw
163 will write to standard output the primary compiler's standard output.
168 will write to standard output the primary compiler's standard output, and the
169 secondary compiler's standard output will be discarded.
164 .Pp
165 Because the Sun compilers write intermediate objects to fixed
166 filenames in the current directory when instructed to compile and
167 link multiple source files via a single command line, it would be
168 unsafe to invoke more than one compiler in this fashion.
169 Therefore
170 .Nm cw
171 does not accept multiple source files unless the preprocessor is to be
172 invoked.
173 An attempt to invoke
174 .Nm cw
175 in this manner will result in an error.
176 .Sh ARGUMENT TRANSLATION
177 If the compiler to be invoked is a GNU-style C or C++ compiler, a set of
178 default flags is added to the beginning of the argument list, and the
179 remaining arguments are translated to their closest appropriate
180 semantic equivalents and passed in the same order as their
181 counterparts given to

```

```

182 .Nm cw .
183 See the comments at the head of
184 .Pa usr/src/tools/cw/cw.c
185 for a detailed list of translations.
186 .Sh ENVIRONMENT
187 .Bl -tag -width indent
188 .It CW_SHADOW_SERIAL
189 If this variable is set in the environment, invoke the primary compiler, wait
190 for it to complete, then invoke the shadow compilers.
191 Normally the primary and shadow compilers are invoked in parallel.
192 .It CW_NO_EXEC
193 If this variable is set in the environment, write the usual output to
194 standard error but do not actually invoke any compiler.
195 This is useful for debugging the translation engine.
196 .El
197 .Sh EXIT STATUS
198 The following exit status values are returned:
199 .Bl -tag -width indent
200 .It 0
201 The primary compiler, and shadow compilers if invoked, both completed
202 successfully.
203 .It >0
204 A usage error occurred, or one or more compilers returned a nonzero
205 exit status.
206 .El
207 .Sh SEE ALSO
208 .Xr cc 1 ,
209 .Xr CC 1 ,
210 .Xr gcc 1
211 .Sh BUGS
212 The translations provided for gcc are not always exact and in some cases
213 reflect local policy rather than actual equivalence.
214 .Pp
215 Additional compiler types should be supported.
216 .Pp
217 The translation engine is hacky.

```

```

*****
45601 Mon Feb 19 00:52:51 2018
new/usr/src/tools/cw/cw.c
cw: don't shadow pure pre-processing
*****
_____unchanged_portion_omitted_____

614 static void
615 do_gcc(cw_ictx_t *ctx)
616 {
617     int c;
618     int pic = 0, nolibc = 0;
619     int in_output = 0, seen_o = 0, c_files = 0;
620     cw_op_t op = CW_O_LINK;
621     char *model = NULL;
622     char *nameflag;
623     int mflag = 0;

625     if (ctx->i_flags & CW_F_PROG) {
626         newae(ctx->i_ae, "--version");
627         return;
628     }

630     newae(ctx->i_ae, "-fident");
631     newae(ctx->i_ae, "-finline");
632     newae(ctx->i_ae, "-fno-inline-functions");
633     newae(ctx->i_ae, "-fno-builtin");
634     newae(ctx->i_ae, "-fno-asm");
635     newae(ctx->i_ae, "-fdiagnostics-show-option");
636     newae(ctx->i_ae, "-nodefaultlibs");

638 #if defined(__sparc)
639     /*
640     * The SPARC ldd and std instructions require 8-byte alignment of
641     * their address operand. gcc correctly uses them only when the
642     * ABI requires 8-byte alignment; unfortunately we have a number of
643     * pieces of buggy code that doesn't conform to the ABI. This
644     * flag makes gcc work more like Studio with -xmalign=4.
645     */
646     newae(ctx->i_ae, "-mno-integer-ldd-std");
647 #endif

649     /*
650     * This is needed because 'u' is defined
651     * under a conditional on 'sun'. Should
652     * probably just remove the conditional,
653     * or make it be dependent on '__sun'.
654     *
655     * -Dunix is also missing in enhanced ANSI mode
656     */
657     newae(ctx->i_ae, "-D__sun");

659     if (asprintf(&nameflag, "-_%s=", ctx->i_compiler->c_name) == -1)
660         nomem();

662     /*
663     * Walk the argument list, translating as we go ..
664     */
665     while (--ctx->i_oldargc > 0) {
666         char *arg = **++ctx->i_oldargv;
667         size_t arglen = strlen(arg);

669         if (*arg == '-') {
670             arglen--;
671         } else {
672             /*

```

```

673     * Discard inline files that gcc doesn't grok
674     */
675     if (!in_output && arglen > 3 &&
676         strcmp(arg + arglen - 3, ".il") == 0)
677         continue;

679     if (!in_output && arglen > 2 &&
680         arg[arglen - 2] == '.' &&
681         (arg[arglen - 1] == 'S' || arg[arglen - 1] == 's' ||
682          arg[arglen - 1] == 'c' || arg[arglen - 1] == 'i'))
683         c_files++;

685     /*
686     * Otherwise, filenames and partial arguments
687     * are passed through for gcc to chew on. However,
688     * output is always discarded for the secondary
689     * compiler.
690     */
691     if ((ctx->i_flags & CW_F_SHADOW) && in_output)
692         newae(ctx->i_ae, ctx->i_discard);
693     else
694         newae(ctx->i_ae, arg);
695     in_output = 0;
696     continue;
697 }

699     if (ctx->i_flags & CW_F_CXX) {
700         if (strncmp(arg, "-g++=", 6) == 0) {
701             newae(ctx->i_ae, strchr(arg, '=') + 1);
702             continue;
703         }
704         if (strncmp(arg, "-compat=", 8) == 0) {
705             /* discard -compat=4 and -compat=5 */
706             continue;
707         }
708         if (strcmp(arg, "-Qoption") == 0) {
709             /* discard -Qoption and its two arguments */
710             if (ctx->i_oldargc < 3)
711                 error(arg);
712             ctx->i_oldargc -= 2;
713             ctx->i_oldargv += 2;
714             continue;
715         }
716         if (strcmp(arg, "-xwe") == 0) {
717             /* turn warnings into errors */
718             newae(ctx->i_ae, "-Werror");
719             continue;
720         }
721         if (strcmp(arg, "-noex") == 0) {
722             /* no exceptions */
723             newae(ctx->i_ae, "-fno-exceptions");
724             /* no run time type descriptor information */
725             newae(ctx->i_ae, "-fno-rtti");
726             continue;
727         }
728         if (strcmp(arg, "-pic") == 0) {
729             newae(ctx->i_ae, "-fpic");
730             pic = 1;
731             continue;
732         }
733         if (strcmp(arg, "-PIC") == 0) {
734             newae(ctx->i_ae, "-fPIC");
735             pic = 1;
736             continue;
737         }
738         if (strcmp(arg, "-norunpath") == 0) {

```

```

739             /* gcc has no corresponding option */
740             continue;
741         }
742         if (strcmp(arg, "-nolib") == 0) {
743             /* -nolib is on by default */
744             nolibc = 1;
745             continue;
746         }
747 #if defined(__sparc)
748         if (strcmp(arg, "-cg92") == 0) {
749             mflag |= xlate_xtb(ctx->i_ae, "v8");
750             xlate(ctx->i_ae, "super", xchip_tbl);
751             continue;
752         }
753 #endif /* __sparc */
754     }

755     switch ((c = arg[1])) {
756     case '_':
757         if ((strcmp(arg, nameflag, strlen(nameflag)) == 0) ||
758             (strcmp(arg, "-_gcc=", 6) == 0) ||
759             (strcmp(arg, "-_gnu=", 6) == 0)) {
760             newae(ctx->i_ae, strchr(arg, '=') + 1);
761         }
762         break;
763     case '#':
764         if (arglen == 1) {
765             newae(ctx->i_ae, "-v");
766             break;
767         }
768         error(arg);
769         break;
770     case 'g':
771         newae(ctx->i_ae, "--gdwarf-2");
772         break;
773     case 'E':
774         if (arglen == 1) {
775             newae(ctx->i_ae, "-xc");
776             newae(ctx->i_ae, arg);
777             op = CW_O_PREPROCESS;
778             nolibc = 1;
779             break;
780         }
781         error(arg);
782         break;
783     case 'c':
784     case 'S':
785         if (arglen == 1) {
786             op = CW_O_COMPILE;
787             nolibc = 1;
788         }
789         /* FALLTHROUGH */
790     case 'C':
791     case 'H':
792     case 'p':
793         if (arglen == 1) {
794             newae(ctx->i_ae, arg);
795             break;
796         }
797         error(arg);
798         break;
799     case 'A':
800     case 'h':
801     case 'I':
802     case 'i':
803     case 'L':

```

```

805     case 'l':
806     case 'R':
807     case 'U':
808     case 'u':
809     case 'w':
810         newae(ctx->i_ae, arg);
811         break;
812     case 'o':
813         seen_o = 1;
814         if (arglen == 1) {
815             in_output = 1;
816             newae(ctx->i_ae, arg);
817         } else if (ctx->i_flags & CW_F_SHADOW) {
818             newae(ctx->i_ae, "-o");
819             newae(ctx->i_ae, ctx->i_discard);
820         } else {
821             newae(ctx->i_ae, arg);
822         }
823         break;
824     case 'D':
825         newae(ctx->i_ae, arg);
826         /*
827          * XXX Clearly a hack ... do we need _KADB too?
828          */
829         if (strcmp(arg, "-D_KERNEL") == 0 ||
830             strcmp(arg, "-D_BOOT") == 0)
831             newae(ctx->i_ae, "-ffreestanding");
832         break;
833     case 'd':
834         if (arglen == 2) {
835             if (strcmp(arg, "-dy") == 0) {
836                 newae(ctx->i_ae, "-Wl,-dy");
837                 break;
838             }
839             if (strcmp(arg, "-dn") == 0) {
840                 newae(ctx->i_ae, "-Wl,-dn");
841                 break;
842             }
843         }
844         if (strcmp(arg, "-dalign") == 0) {
845             /*
846              * -dalign forces alignment in some cases;
847              * gcc does not need any flag to do this.
848              */
849             break;
850         }
851         error(arg);
852         break;
853     case 'e':
854         if (strcmp(arg,
855             "-erroff=E_EMPTY_TRANSLATION_UNIT") == 0) {
856             /*
857              * Accept but ignore this -- gcc doesn't
858              * seem to complain about empty translation
859              * units
860              */
861             break;
862         }
863         /* XX64 -- ignore all -erroff= options, for now */
864         if (strcmp(arg, "-erroff=", 8) == 0)
865             break;
866         if (strcmp(arg, "-errtags=yes") == 0) {
867             warnings(ctx->i_ae);
868             break;
869         }
870         if (strcmp(arg, "-errwarn=%all") == 0) {

```

```

871         newae(ctx->i_ae, "-Werror");
872         break;
873     }
874     error(arg);
875     break;
876 case 'f':
877     if (strcmp(arg, "-flags") == 0) {
878         newae(ctx->i_ae, "--help");
879         break;
880     }
881     if (strncmp(arg, "--features=zla", 13) == 0) {
882         /*
883          * Accept but ignore this -- gcc allows
884          * zero length arrays.
885          */
886         break;
887     }
888     error(arg);
889     break;
890 case 'G':
891     newae(ctx->i_ae, "--shared");
892     nolibc = 1;
893     break;
894 case 'k':
895     if (strcmp(arg, "-keeptmp") == 0) {
896         newae(ctx->i_ae, "-save-temps");
897         break;
898     }
899     error(arg);
900     break;
901 case 'K':
902     if (arglen == 1) {
903         if ((arg == *++ctx->i_oldargv) == NULL ||
904             *arg == '\0')
905             error("-K");
906         ctx->i_oldargc--;
907     } else {
908         arg += 2;
909     }
910     if (strcmp(arg, "pic") == 0) {
911         newae(ctx->i_ae, "-fpic");
912         pic = 1;
913         break;
914     }
915     if (strcmp(arg, "PIC") == 0) {
916         newae(ctx->i_ae, "-fPIC");
917         pic = 1;
918         break;
919     }
920     error("-K");
921     break;
922 case 'm':
923     if (strcmp(arg, "-mt") == 0) {
924         newae(ctx->i_ae, "-D_REENTRANT");
925         break;
926     }
927     if (strcmp(arg, "-m64") == 0) {
928         newae(ctx->i_ae, "-m64");
929 #if defined(__x86)
930         newae(ctx->i_ae, "-mtune=opteron");
931 #endif
932         mflag |= M64;
933         break;
934     }
935     if (strcmp(arg, "-m32") == 0) {
936         newae(ctx->i_ae, "-m32");

```

```

937         mflag |= M32;
938         break;
939     }
940     error(arg);
941     break;
942 case 'B': /* linker options */
943 case 'M':
944 case 'z':
945     {
946         char *opt;
947         size_t len;
948         char *s;
949
950         if (arglen == 1) {
951             opt = *++ctx->i_oldargv;
952             if (opt == NULL || *opt == '\0')
953                 error(arg);
954             ctx->i_oldargc--;
955         } else {
956             opt = arg + 2;
957         }
958         len = strlen(opt) + 7;
959         if ((s = malloc(len)) == NULL)
960             nomem();
961         (void) snprintf(s, len, "-Wl,-%c%s", c, opt);
962         newae(ctx->i_ae, s);
963         free(s);
964     }
965     break;
966 case 'n':
967     if (strcmp(arg, "-noqueue") == 0) {
968         /*
969          * Horrid license server stuff - n/a
970          */
971         break;
972     }
973     error(arg);
974     break;
975 case 'O':
976     if (arglen == 1) {
977         newae(ctx->i_ae, "-O");
978         break;
979     }
980     error(arg);
981     break;
982 case 'P':
983     /*
984      * We could do '-E -o filename.i', but that's hard,
985      * and we don't need it for the case that's triggering
986      * this addition. We'll require the user to specify
987      * -o in the Makefile. If they don't they'll find out
988      * in a hurry.
989      */
990     newae(ctx->i_ae, "-E");
991     op = CW_O_PREPROCESS;
992     nolibc = 1;
993     break;
994 case 'q':
995     if (strcmp(arg, "-qp") == 0) {
996         newae(ctx->i_ae, "-p");
997         break;
998     }
999     error(arg);
1000     break;
1001 case 's':
1002     if (arglen == 1) {

```

```

1003             newae(ctx->i_ae, "-Wl,-s");
1004             break;
1005         }
1006         error(arg);
1007         break;
1008     case 't':
1009         if (arglen == 1) {
1010             newae(ctx->i_ae, "-Wl,-t");
1011             break;
1012         }
1013         error(arg);
1014         break;
1015     case 'V':
1016         if (arglen == 1) {
1017             ctx->i_flags &= ~CW_F_ECHO;
1018             newae(ctx->i_ae, "--version");
1019             break;
1020         }
1021         error(arg);
1022         break;
1023     case 'v':
1024         if (arglen == 1) {
1025             warnings(ctx->i_ae);
1026             break;
1027         }
1028         error(arg);
1029         break;
1030     case 'W':
1031         if (strncmp(arg, "-Wp,-xc99", 9) == 0) {
1032             /*
1033              * gcc's preprocessor will accept c99
1034              * regardless, so accept and ignore.
1035              */
1036             break;
1037         }
1038         if (strncmp(arg, "-Wa,", 4) == 0 ||
1039             strncmp(arg, "-Wp,", 4) == 0 ||
1040             strncmp(arg, "-Wl,", 4) == 0) {
1041             newae(ctx->i_ae, arg);
1042             break;
1043         }
1044         if (strcmp(arg, "-W0,-xc99=pragma") == 0) {
1045             /* (undocumented) enables _Pragma */
1046             break;
1047         }
1048         if (strcmp(arg, "-W0,-xc99=%none") == 0) {
1049             /*
1050              * This is a polite way of saying
1051              * "no c99 constructs allowed!"
1052              * For now, just accept and ignore this.
1053              */
1054             break;
1055         }
1056         if (strcmp(arg, "-W0,-noglobal") == 0 ||
1057             strcmp(arg, "-W0,-xglobalstatic") == 0) {
1058             /*
1059              * gcc doesn't prefix local symbols
1060              * in debug mode, so this is not needed.
1061              */
1062             break;
1063         }
1064         if (strcmp(arg, "-W0,-Lt") == 0) {
1065             /*
1066              * Generate tests at the top of loops.
1067              * There is no direct gcc equivalent, ignore.
1068              */

```

```

1069             break;
1070         }
1071         if (strcmp(arg, "-W0,-xdbggen=no%usedonly") == 0) {
1072             newae(ctx->i_ae,
1073                 "-fno-eliminate-unused-debug-symbols");
1074             newae(ctx->i_ae,
1075                 "-fno-eliminate-unused-debug-types");
1076             break;
1077         }
1078         if (strcmp(arg, "-W2,-xwrap_int") == 0) {
1079             /*
1080              * Use the legacy behaviour (pre-SS11)
1081              * for integer wrapping.
1082              * gcc does not need this.
1083              */
1084             break;
1085         }
1086         if (strcmp(arg, "-W2,-Rcond_elim") == 0) {
1087             /*
1088              * Elimination and expansion of conditionals;
1089              * gcc has no direct equivalent.
1090              */
1091             break;
1092         }
1093         if (strcmp(arg, "-Wd,-xsafe=unboundsym") == 0) {
1094             /*
1095              * Prevents optimizing away checks for
1096              * unbound weak symbol addresses. gcc does
1097              * not do this, so it's not needed.
1098              */
1099             break;
1100         }
1101         if (strncmp(arg, "-Wc,-xcode=", 11) == 0) {
1102             xlate(ctx->i_ae, arg + 11, xcode_tbl);
1103             if (strncmp(arg + 11, "pic", 3) == 0)
1104                 pic = 1;
1105             break;
1106         }
1107         if (strncmp(arg, "-Wc,-Qiselect", 13) == 0) {
1108             /*
1109              * Prevents insertion of register symbols.
1110              * gcc doesn't do this, so ignore it.
1111              */
1112             break;
1113         }
1114         if (strcmp(arg, "-Wc,-Qassembler-ounrefsym=0") == 0) {
1115             /*
1116              * Prevents optimizing away of static variables.
1117              * gcc does not do this, so it's not needed.
1118              */
1119             break;
1120         }
1121         #if defined(__x86)
1122         if (strcmp(arg, "-Wu,-xmodel=kernel") == 0) {
1123             newae(ctx->i_ae, "-ffreestanding");
1124             newae(ctx->i_ae, "-mno-red-zone");
1125             model = "mcmmodel=kernel";
1126             nolibc = 1;
1127             break;
1128         }
1129         if (strcmp(arg, "-Wu,-save_args") == 0) {
1130             newae(ctx->i_ae, "-msave_args");
1131             break;
1132         }
1133         #endif /* __x86 */
1134         error(arg);

```

```

1135         break;
1136     case 'X':
1137         if (strcmp(arg, "-Xa") == 0 ||
1138             strcmp(arg, "-Xt") == 0) {
1139             Xamode(ctx->i_ae);
1140             break;
1141         }
1142         if (strcmp(arg, "-Xc") == 0) {
1143             Xcmode(ctx->i_ae);
1144             break;
1145         }
1146         if (strcmp(arg, "-Xs") == 0) {
1147             Xsmode(ctx->i_ae);
1148             break;
1149         }
1150         error(arg);
1151         break;
1152     case 'x':
1153         if (arglen == 1)
1154             error(arg);
1155         switch (arg[2]) {
1156 #if defined(__x86)
1157         case '3':
1158             if (strcmp(arg, "-x386") == 0) {
1159                 newae(ctx->i_ae, "-march=i386");
1160                 break;
1161             }
1162             error(arg);
1163             break;
1164         case '4':
1165             if (strcmp(arg, "-x486") == 0) {
1166                 newae(ctx->i_ae, "-march=i486");
1167                 break;
1168             }
1169             error(arg);
1170             break;
1171 #endif /* __x86 */
1172         case 'a':
1173             if (strcmp(arg, "-xarch=", 7) == 0) {
1174                 mflag |= xlate_xtb(ctx->i_ae, arg + 7);
1175                 break;
1176             }
1177             error(arg);
1178             break;
1179         case 'b':
1180             if (strcmp(arg, "-xbuiltin=", 10) == 0) {
1181                 if (strcmp(arg + 10, "%all") == 0)
1182                     newae(ctx->i_ae, "-fbuiltin");
1183                 break;
1184             }
1185             error(arg);
1186             break;
1187         case 'C':
1188             /* Accept C++ style comments -- ignore */
1189             if (strcmp(arg, "-xCC") == 0)
1190                 break;
1191             error(arg);
1192             break;
1193         case 'c':
1194             if (strcmp(arg, "-xc99=%all", 10) == 0) {
1195                 newae(ctx->i_ae, "-std=gnu99");
1196                 break;
1197             }
1198             if (strcmp(arg, "-xc99=%none", 11) == 0) {
1199                 newae(ctx->i_ae, "-std=gnu89");
1200                 break;

```

```

1201     }
1202     if (strcmp(arg, "-xchip=", 7) == 0) {
1203         xlate(ctx->i_ae, arg + 7, xchip_tbl);
1204         break;
1205     }
1206     if (strcmp(arg, "-xcode=", 7) == 0) {
1207         xlate(ctx->i_ae, arg + 7, xcode_tbl);
1208         if (strcmp(arg + 7, "pic", 3) == 0)
1209             pic = 1;
1210         break;
1211     }
1212     if (strcmp(arg, "-xcache=", 8) == 0)
1213         break;
1214     if (strcmp(arg, "-xcrossfile", 11) == 0)
1215         break;
1216     error(arg);
1217     break;
1218 case 'd':
1219     if (strcmp(arg, "-xdepend") == 0)
1220         break;
1221     if (strcmp(arg, "-xdebugformat=", 14) == 0)
1222         break;
1223     error(arg);
1224     break;
1225 case 'F':
1226     /*
1227      * Compile for mapfile reordering, or unused
1228      * section elimination, syntax can be -xF or
1229      * more complex, like -xF=%all -- ignore.
1230      */
1231     if (strcmp(arg, "-xF", 3) == 0)
1232         break;
1233     error(arg);
1234     break;
1235 case 'i':
1236     if (strcmp(arg, "-xinline", 8) == 0)
1237         /* No inlining; ignore */
1238         break;
1239     if (strcmp(arg, "-xildon") == 0 ||
1240         strcmp(arg, "-xildoff") == 0)
1241         /* No incremental linking; ignore */
1242         break;
1243     error(arg);
1244     break;
1245 #if defined(__x86)
1246 case 'm':
1247     if (strcmp(arg, "-xmodel=kernel") == 0) {
1248         newae(ctx->i_ae, "-ffreestanding");
1249         newae(ctx->i_ae, "-mno-red-zone");
1250         model = "-mcmmodel=kernel";
1251         nolibc = 1;
1252         break;
1253     }
1254     error(arg);
1255     break;
1256 #endif /* __x86 */
1257 case 'M':
1258     if (strcmp(arg, "-xM") == 0) {
1259         newae(ctx->i_ae, "-M");
1260         break;
1261     }
1262     if (strcmp(arg, "-xM1") == 0) {
1263         newae(ctx->i_ae, "-MM");
1264         break;
1265     }
1266     error(arg);

```

```

1267         break;
1268     case 'n':
1269         if (strcmp(arg, "-xnoilib") == 0) {
1270             nolibc = 1;
1271             break;
1272         }
1273         error(arg);
1274         break;
1275     case 'O':
1276         if (strncmp(arg, "-xO", 3) == 0) {
1277             size_t len = strlen(arg);
1278             char *s;
1279             int c = *(arg + 3);
1280             int level;
1281
1282             if (len != 4 || !isdigit(c))
1283                 error(arg);
1284
1285             if ((s = malloc(len)) == NULL)
1286                 nomem();
1287
1288             level = atoi(arg + 3);
1289             if (level > 5)
1290                 error(arg);
1291             if (level >= 2) {
1292                 /*
1293                  * For gcc-3.4.x at -O2 we
1294                  * need to disable optimizations
1295                  * that break ON.
1296                  */
1297                 optim_disable(ctx->i_ae, level);
1298                 /*
1299                  * limit -xO3 to -O2 as well.
1300                  */
1301                 level = 2;
1302             }
1303             (void) snprintf(s, len, "-O%d", level);
1304             newae(ctx->i_ae, s);
1305             free(s);
1306             break;
1307         }
1308         error(arg);
1309         break;
1310     case 'p':
1311         if (strcmp(arg, "-xpentium") == 0) {
1312             newae(ctx->i_ae, "-march=pentium");
1313             break;
1314         }
1315         if (strcmp(arg, "-xpg") == 0) {
1316             newae(ctx->i_ae, "-pg");
1317             break;
1318         }
1319         error(arg);
1320         break;
1321     case 'r':
1322         if (strncmp(arg, "-xregs=", 7) == 0) {
1323             xlate(ctx->i_ae, arg + 7, xregs_tbl);
1324             break;
1325         }
1326         error(arg);
1327         break;
1328     case 's':
1329         if (strcmp(arg, "-xs") == 0 ||
1330             strcmp(arg, "-xspace") == 0 ||
1331             strcmp(arg, "-xstrconst") == 0)
1332             break;

```

```

1333         error(arg);
1334         break;
1335     case 't':
1336         if (strcmp(arg, "-xtransition") == 0) {
1337             newae(ctx->i_ae, "-Wtransition");
1338             break;
1339         }
1340         if (strcmp(arg, "-xtrigraphs=yes") == 0) {
1341             newae(ctx->i_ae, "-trigraphs");
1342             break;
1343         }
1344         if (strcmp(arg, "-xtrigraphs=no") == 0) {
1345             newae(ctx->i_ae, "-notrigraphs");
1346             break;
1347         }
1348         if (strncmp(arg, "-xtarget=", 9) == 0) {
1349             xlate(ctx->i_ae, arg + 9, xtarget_tbl);
1350             break;
1351         }
1352         error(arg);
1353         break;
1354     case 'e':
1355     case 'h':
1356     case 'l':
1357     default:
1358         error(arg);
1359         break;
1360     }
1361     break;
1362 case 'Y':
1363     if (arglen == 1) {
1364         if ((arg = **+ctx->i_oldargv) == NULL ||
1365             *arg == '\0')
1366             error("-Y");
1367         ctx->i_oldargc--;
1368         arglen = strlen(arg + 1);
1369     } else {
1370         arg += 2;
1371     }
1372     /* Just ignore -YS,... for now */
1373     if (strncmp(arg, "S,", 2) == 0)
1374         break;
1375     if (strncmp(arg, "l,", 2) == 0) {
1376         char *s = strdup(arg);
1377         s[0] = '-';
1378         s[1] = 'B';
1379         newae(ctx->i_ae, s);
1380         free(s);
1381         break;
1382     }
1383     if (strncmp(arg, "I,", 2) == 0) {
1384         char *s = strdup(arg);
1385         s[0] = '-';
1386         s[1] = 'I';
1387         newae(ctx->i_ae, "-nostdinc");
1388         newae(ctx->i_ae, s);
1389         free(s);
1390         break;
1391     }
1392     error(arg);
1393     break;
1394 case 'Q':
1395     /*
1396      * We could map -Qy into -Wl,-Qy etc.
1397      */
1398     default:

```



```

1399         error(arg);
1400         break;
1401     }
1402 }

1404 free(nameflag);

1406 if (c_files > 1 && (ctx->i_flags & CW_F_SHADOW) &&
1407     op != CW_O_PREPROCESS) {
1408     errx(2, "multiple source files are "
1409         "allowed only with -E or -P");
1410 }

1412 /*
1413  * Make sure that we do not have any unintended interactions between
1414  * the xarch options passed in and the version of the Studio compiler
1415  * used.
1416  */
1417 if ((mflag & (SS11|SS12)) == (SS11|SS12)) {
1418     errx(2,
1419         "Conflicting \"-xarch=\" flags (both Studio 11 and 12)\n");
1420 }

1422 switch (mflag) {
1423 case 0:
1424     /* FALLTHROUGH */
1425     case M32:
1426 #if defined(__sparc)
1427     /*
1428      * Only -m32 is defined and so put in the missing xarch
1429      * translation.
1430      */
1431     newae(ctx->i_ae, "-mcpu=v8");
1432     newae(ctx->i_ae, "-mno-v8plus");
1433 #endif
1434     break;
1435     case M64:
1436 #if defined(__sparc)
1437     /*
1438      * Only -m64 is defined and so put in the missing xarch
1439      * translation.
1440      */
1441     newae(ctx->i_ae, "-mcpu=v9");
1442 #endif
1443     break;
1444     case SS12:
1445 #if defined(__sparc)
1446     /* no -m32/-m64 flag used - this is an error for sparc builds */
1447     (void) fprintf(stderr, "No -m32/-m64 flag defined\n");
1448     exit(2);
1449 #endif
1450     break;
1451     case SS11:
1452     /* FALLTHROUGH */
1453     case (SS11|M32):
1454     case (SS11|M64):
1455     break;
1456     case (SS12|M32):
1457 #if defined(__sparc)
1458     /*
1459      * Need to add in further 32 bit options because with SS12
1460      * the xarch=sparcvis option can be applied to 32 or 64
1461      * bit, and so the translation table (xtbl) cannot handle
1462      * that.
1463      */
1464     newae(ctx->i_ae, "-mv8plus");

```

```

1465 #endif
1466         break;
1467     case (SS12|M64):
1468     break;
1469     default:
1470     (void) fprintf(stderr,
1471         "Incompatible -xarch= and/or -m32/-m64 options used.\n");
1472     exit(2);
1473 }

1475 if ((op == CW_O_LINK || op == CW_O_PREPROCESS) &&
1476     (ctx->i_flags & CW_F_SHADOW))
1477 if (op == CW_O_LINK && (ctx->i_flags & CW_F_SHADOW))
1478     exit(0);

1479 if (model && !pic)
1480     newae(ctx->i_ae, model);
1481 if (!nolibc)
1482     newae(ctx->i_ae, "-lc");
1483 if (!seen_o && (ctx->i_flags & CW_F_SHADOW)) {
1484     newae(ctx->i_ae, "-o");
1485     newae(ctx->i_ae, ctx->i_discard);
1486 }
1487 }

unchanged_portion_omitted

```