

```
*****
46122 Mon Nov  5 20:40:46 2012
new/usr/src/cmd/sgs/libld/common/machrel.amd.c
3337 x64 link-editor is painfully literal-minded about TLS
*****  

_____ unchanged_portion_omitted _____
```

```

517 #define REX_B          0x1
518 #define REX_X          0x2
519 #define REX_R          0x4
520 #define REX_W          0x8
521 #define REX_PREFIX      0x40

523 #define REX_RW          (REX_PREFIX | REX_R | REX_W)
524 #define REX_BW          (REX_PREFIX | REX_B | REX_W)
525 #define REX_BRW         (REX_PREFIX | REX_B | REX_R | REX_W)

527 #define REG_ESP         0x4

529 #define INSN_ADDMR      0x03 /* addq mem,reg */
530 #define INSN_ADDIR      0x81 /* addq imm,reg */
531 #define INSN_MOVMR      0x8b /* movq mem,reg */
532 #define INSN_MOVIR      0xc7 /* movq imm,reg */
533 #define INSN_LEA          0x8d /* leaq mem,reg */
534 #endif /* ! codereview */

536 static Fixupret
537 tls_fixups(Ofl_desc *ofl, Rel_desc *arsp)
538 {
539     Sym_desc      *sdp = arsp->rel_sym;
540     Word           rtype = arsp->rel_rtype;
541     uchar_t        *offset;

543     offset = (uchar_t *)((uintptr_t)arsp->rel_roffset +
544                           (uintptr_t)_elf_getxoff(arsp->rel_isdesc->is_indata) +
545                           (uintptr_t)RELAUX_GET_OSDESC(arsp)->os_outdata->d_buf);

547     /*
548      * Note that in certain of the original insn sequences below, the
549      * instructions are not necessarily adjacent
550     */
551 #endif /* ! codereview */
552     if (sdp->sdp_ref == REF_DYN_NEED) {
553         /*
554          * IE reference model
555         */
556         switch (rtype) {
557             case R_AMD64_TLSDG:
558                 /*
559                  * GD -> IE
560                  *
561                  * Transition:
562                  *   0x00 .byte 0x66
563                  *   0x01 leaq x@tlsqd(%rip), %rdi
564                  *   0x08 .word 0x6666
565                  *   0x0a rex64
566                  *   0x0b call __tls_get_addr@plt
567                  *   0x10
568                  *
569                  * To:
570                  *   0x00 movq %fs:0, %rax
571                  *   0x09 addq x@gottpoff(%rip), %rax
572                  *   0x10
573                  */
574                 DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
575                                              R_AMD64_GOTTPOFF, arsp, ld_reloc_sym_name));
576                 arsp->rel_rtype = R_AMD64_GOTTPOFF;
```

```

576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
```

```

    arsp->rel_roffset += 8;
    arsp->rel_raddend = (Sxword)-4;

    /*
     * Adjust 'offset' to beginning of instruction
     * sequence.
     */
    offset -= 4;
    (void) memcpy(offset, tlsinstr_gd_ie,
                 sizeof(tlsinstr_gd_ie));
    return (FIX_RELOC);

case R_AMD64_PLT32:
    /*
     * Fixup done via the TLS_GD relocation.
     */
    DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
                                  R_AMD64_NONE, arsp, ld_reloc_sym_name));
    return (FIX_DONE);
}

/*
 * LE reference model
 */
switch (rtype) {
case R_AMD64_TLSDG:
    /*
     * GD -> LE
     *
     * Transition:
     *   0x00 .byte 0x66
     *   0x01 leaq x@tlsqd(%rip), %rdi
     *   0x08 .word 0x6666
     *   0x0a rex64
     *   0x0b call __tls_get_addr@plt
     *   0x10
     *
     * To:
     *   0x00 movq %fs:0, %rax
     *   0x09 leaq x@tpoff(%rax), %rax
     *   0x10
     */
    DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
                                  R_AMD64_TPOFF32, arsp, ld_reloc_sym_name));
    arsp->rel_rtype = R_AMD64_TPOFF32;
    arsp->rel_roffset += 8;
    arsp->rel_raddend = 0;

    /*
     * Adjust 'offset' to beginning of instruction sequence.
     */
    offset -= 4;
    (void) memcpy(offset, tlsinstr_gd_le, sizeof(tlsinstr_gd_le));
    return (FIX_RELOC);

case R_AMD64_GOTTPOFF: {
case R_AMD64_GOTTPOFF:
    /*
     * IE -> LE
     *
     * Transition 1:
     *   movq %fs:0, %reg
     *   addq x@gottpoff(%rip), %reg
     * To:
     *   movq %fs:0, %reg
     *   leaq x@tpoff(%reg), %reg
     */
}
```

```

641           *
642           * Transition (as a special case):
643           *      movq %fs:0, %r12/%rsp
644           *      addq x@gottpoff(%rip), %r12/%rsp
645           * To:
646           *      movq %fs:0, %r12/%rsp
647           *      addq x@tpoff(%rax), %r12/%rsp
648           *
649           * Transition 2:
650           *      movq x@gottpoff(%rip), %reg
651           *      movq %fs:(%reg), %reg
652           * Transition:
653           *      0x00 movq %fs:0, %rax
654           *      0x09 addq x@gottopoff(%rip), %rax
655           *      0x10
656           * To:
657           *      movq x@tpoff(%reg), %reg
658           *      movq %fs:(%reg), %reg
659           *      0x00 movq %fs:0, %rax
660           *      0x09 leaq x@tpoff(%rax), %rax
661           *      0x10
662           */
663     Conv_inv_buf_t inv_buf;
664     uint8_t reg;          /* Register */
665
666     offset -= 3;
667
668     reg = offset[2] >> 3; /* Encoded dest. reg. operand */
669
670 #endif /* ! codereview */
671     DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
672                                   R_AMD64_TPOFF32, arsp, ld_reloc_sym_name));
673     arsp->rel_rtype = R_AMD64_TPOFF32;
674     arsp->rel_raddend = 0;
675
676     /*
677     * This is transition 2, and the special case of form 1 where
678     * a normal transition would index %rsp or %r12 and need a SIB
679     * byte in the leaq for which we lack space
680     */
681     if ((offset[1] == INSN_MOVMR) ||
682         ((offset[1] == INSN_ADDMR) && (reg == REG_ESP))) {
683         /*
684         * If we needed an extra bit of MOD.reg to refer to
685         * this register as the dest of the original movq we
686         * need an extra bit of MOD.rm to refer to it in the
687         * dest of the replacement movq or addq.
688         */
689         if (offset[0] == REX_RW)
690             offset[0] = REX_BW;
691
692         offset[1] = (offset[1] == INSN_MOVMR) ?
693                     INSN_MOVIR : INSN_ADDIR;
694         offset[2] = 0xc0 | reg;
695
696         /* Adjust 'offset' to beginning of instruction sequence.
697         */
698         offset -= 12;
699
700         return (FIX_RELOC);
701     } else if (offset[1] == INSN_ADDMR) {
702
703         /*
704         * If we needed an extra bit of MOD.reg to refer to
705         * this register in the dest of the addq we need an
706         * extra bit of both MOD.reg and MOD.rm to refer to it
707         * in the source and dest of the leaq
708
709     }
710
711 #endif /* ! codereview */
712
713     /* If we needed an extra bit of MOD.reg to refer to
714     * this register in the dest of the addq we need an
715     * extra bit of both MOD.reg and MOD.rm to refer to it
716
717     */
718
719     /* LD -> LE
720
721     * Transition
722     *      0x00 leaq x1@tlsgd(%rip), %rdi
723     *      0x07 call __tls_get_addr@plt
724     *      0x0c
725     * To:
726     *      0x00 .byte 0x66
727     *      0x01 .byte 0x66
728     *      0x02 .byte 0x66
729     *      0x03 movq %fs:0, %rax
730
731     */
732     DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
733                                   R_AMD64_NONE, arsp, ld_reloc_sym_name));
734     offset -= 3;
735     (void) memcpy(offset, tlsinstr_ld_le, sizeof(tlsinstr_ld_le));
736     return (FIX_DONE);
737
738     /* LD->LE
739
740     * Transition:
741     *      0x00 leaq x1@dtloff(%rax), %rcx
742     * To:
743     *      0x00 leaq x1@tpoff(%rax), %rcx
744
745     */
746     DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
747                                   R_AMD64_TPOFF32, arsp, ld_reloc_sym_name));
748     arsp->rel_rtype = R_AMD64_TPOFF32;
749     arsp->rel_raddend = 0;
750     return (FIX_RELOC);
751
752
753     return (FIX_RELOC);
754
755     static uintptr_t
756     ld_do_activerelocs(Ofl_desc *ofl)
757     {
758         Rel_desc        *arsp;
759         Rel_cachebuf   *rcbp;
760
761     }
762
763     /* Same code sequence used in the GD -> LE transition.
764
765     */
766     if (offset[0] == REX_RW)
767         offset[0] = REX_BW;
768
769     offset[1] = INSN_LEA;
770     offset[2] = 0x80 | (reg << 3) | reg;
771
772     (void) memcpy(offset, tlsinstr_gd_le, sizeof(tlsinstr_gd_le));
773     return (FIX_RELOC);
774
775 #endif /* ! codereview */
776
777     ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_BADTLSINS),
778                conv_reloc_amd64_type(arsp->rel_rtype, 0, &inv_buf),
779                arsp->rel_isdesc->is_file->ifl_name,
780                ld_reloc_sym_name(arsp),
781                arsp->rel_isdesc->is_name,
782                EC_OFF(arsp->rel_roffset));
783     return (FIX_ERROR);
784
785 }
786
787 #endif /* ! codereview */
788
789     /* LD -> LE
790
791     * Transition
792     *      0x00 leaq x1@tlsgd(%rip), %rdi
793     *      0x07 call __tls_get_addr@plt
794     *      0x0c
795     * To:
796     *      0x00 .byte 0x66
797     *      0x01 .byte 0x66
798     *      0x02 .byte 0x66
799     *      0x03 movq %fs:0, %rax
800
801     */
802     DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
803                                   R_AMD64_NONE, arsp, ld_reloc_sym_name));
804     offset -= 3;
805     (void) memcpy(offset, tlsinstr_ld_le, sizeof(tlsinstr_ld_le));
806     return (FIX_DONE);
807
808     /* LD->LE
809
810     * Transition:
811     *      0x00 leaq x1@dtloff(%rax), %rcx
812     * To:
813     *      0x00 leaq x1@tpoff(%rax), %rcx
814
815     */
816     DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
817                                   R_AMD64_TPOFF32, arsp, ld_reloc_sym_name));
818     arsp->rel_rtype = R_AMD64_TPOFF32;
819     arsp->rel_raddend = 0;
820     return (FIX_RELOC);
821
822
823     static uintptr_t
824     ld_do_activerelocs(Ofl_desc *ofl)
825     {
826         Rel_desc        *arsp;
827         Rel_cachebuf   *rcbp;
828
829     }
830
831     /* Same code sequence used in the GD -> LE transition.
832
833     */
834     if (offset[0] == REX_RW)
835         offset[0] = REX_BW;
836
837     offset[1] = INSN_LEA;
838     offset[2] = 0x80 | (reg << 3) | reg;
839
840     (void) memcpy(offset, tlsinstr_gd_le, sizeof(tlsinstr_gd_le));
841     return (FIX_RELOC);
842
843 #endif /* ! codereview */
844
845     ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_BADTLSINS),
846                conv_reloc_amd64_type(arsp->rel_rtype, 0, &inv_buf),
847                arsp->rel_isdesc->is_file->ifl_name,
848                ld_reloc_sym_name(arsp),
849                arsp->rel_isdesc->is_name,
850                EC_OFF(arsp->rel_roffset));
851     return (FIX_ERROR);
852
853 }
854
855 #endif /* ! codereview */
856
857     /* LD -> LE
858
859     * Transition
860     *      0x00 leaq x1@tlsgd(%rip), %rdi
861     *      0x07 call __tls_get_addr@plt
862     *      0x0c
863     * To:
864     *      0x00 .byte 0x66
865     *      0x01 .byte 0x66
866     *      0x02 .byte 0x66
867     *      0x03 movq %fs:0, %rax
868
869     */
870     DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
871                                   R_AMD64_NONE, arsp, ld_reloc_sym_name));
872     offset -= 3;
873     (void) memcpy(offset, tlsinstr_ld_le, sizeof(tlsinstr_ld_le));
874     return (FIX_DONE);
875
876     /* LD->LE
877
878     * Transition:
879     *      0x00 leaq x1@dtloff(%rax), %rcx
880     * To:
881     *      0x00 leaq x1@tpoff(%rax), %rcx
882
883     */
884     DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
885                                   R_AMD64_TPOFF32, arsp, ld_reloc_sym_name));
886     arsp->rel_rtype = R_AMD64_TPOFF32;
887     arsp->rel_raddend = 0;
888     return (FIX_RELOC);
889
890
891     static uintptr_t
892     ld_do_activerelocs(Ofl_desc *ofl)
893     {
894         Rel_desc        *arsp;
895         Rel_cachebuf   *rcbp;
896
897     }
898
899     /* Same code sequence used in the GD -> LE transition.
900
901     */
902     if (offset[0] == REX_RW)
903         offset[0] = REX_BW;
904
905     offset[1] = INSN_LEA;
906     offset[2] = 0x80 | (reg << 3) | reg;
907
908     (void) memcpy(offset, tlsinstr_gd_le, sizeof(tlsinstr_gd_le));
909     return (FIX_RELOC);
910
911 #endif /* ! codereview */
912
913     ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_BADTLSINS),
914                conv_reloc_amd64_type(arsp->rel_rtype, 0, &inv_buf),
915                arsp->rel_isdesc->is_file->ifl_name,
916                ld_reloc_sym_name(arsp),
917                arsp->rel_isdesc->is_name,
918                EC_OFF(arsp->rel_roffset));
919     return (FIX_ERROR);
920
921 }
922
923 #endif /* ! codereview */
924
925     /* LD -> LE
926
927     * Transition
928     *      0x00 leaq x1@tlsgd(%rip), %rdi
929     *      0x07 call __tls_get_addr@plt
930     *      0x0c
931     * To:
932     *      0x00 .byte 0x66
933     *      0x01 .byte 0x66
934     *      0x02 .byte 0x66
935     *      0x03 movq %fs:0, %rax
936
937     */
938     DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
939                                   R_AMD64_NONE, arsp, ld_reloc_sym_name));
940     offset -= 3;
941     (void) memcpy(offset, tlsinstr_ld_le, sizeof(tlsinstr_ld_le));
942     return (FIX_DONE);
943
944     /* LD->LE
945
946     * Transition:
947     *      0x00 leaq x1@dtloff(%rax), %rcx
948     * To:
949     *      0x00 leaq x1@tpoff(%rax), %rcx
950
951     */
952     DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
953                                   R_AMD64_TPOFF32, arsp, ld_reloc_sym_name));
954     arsp->rel_rtype = R_AMD64_TPOFF32;
955     arsp->rel_raddend = 0;
956     return (FIX_RELOC);
957
958
959     static uintptr_t
960     ld_do_activerelocs(Ofl_desc *ofl)
961     {
962         Rel_desc        *arsp;
963         Rel_cachebuf   *rcbp;
964
965     }
966
967     /* Same code sequence used in the GD -> LE transition.
968
969     */
970     if (offset[0] == REX_RW)
971         offset[0] = REX_BW;
972
973     offset[1] = INSN_LEA;
974     offset[2] = 0x80 | (reg << 3) | reg;
975
976     (void) memcpy(offset, tlsinstr_gd_le, sizeof(tlsinstr_gd_le));
977     return (FIX_RELOC);
978
979 #endif /* ! codereview */
980
981     ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_BADTLSINS),
982                conv_reloc_amd64_type(arsp->rel_rtype, 0, &inv_buf),
983                arsp->rel_isdesc->is_file->ifl_name,
984                ld_reloc_sym_name(arsp),
985                arsp->rel_isdesc->is_name,
986                EC_OFF(arsp->rel_roffset));
987     return (FIX_ERROR);
988
989 }
990
991 #endif /* ! codereview */
992
993     /* LD -> LE
994
995     * Transition
996     *      0x00 leaq x1@tlsgd(%rip), %rdi
997     *      0x07 call __tls_get_addr@plt
998     *      0x0c
999     * To:
1000     *      0x00 .byte 0x66
1001     *      0x01 .byte 0x66
1002     *      0x02 .byte 0x66
1003     *      0x03 movq %fs:0, %rax
1004
1005     */
1006     DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
1007                                   R_AMD64_NONE, arsp, ld_reloc_sym_name));
1008     offset -= 3;
1009     (void) memcpy(offset, tlsinstr_ld_le, sizeof(tlsinstr_ld_le));
1010     return (FIX_DONE);
1011
1012     /* LD->LE
1013
1014     * Transition:
1015     *      0x00 leaq x1@dtloff(%rax), %rcx
1016     * To:
1017     *      0x00 leaq x1@tpoff(%rax), %rcx
1018
1019     */
1020     DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
1021                                   R_AMD64_TPOFF32, arsp, ld_reloc_sym_name));
1022     arsp->rel_rtype = R_AMD64_TPOFF32;
1023     arsp->rel_raddend = 0;
1024     return (FIX_RELOC);
1025
1026
1027     static uintptr_t
1028     ld_do_activerelocs(Ofl_desc *ofl)
1029     {
1030         Rel_desc        *arsp;
1031         Rel_cachebuf   *rcbp;
1032
1033     }
1034
1035     /* Same code sequence used in the GD -> LE transition.
1036
1037     */
1038     if (offset[0] == REX_RW)
1039         offset[0] = REX_BW;
1040
1041     offset[1] = INSN_LEA;
1042     offset[2] = 0x80 | (reg << 3) | reg;
1043
1044     (void) memcpy(offset, tlsinstr_gd_le, sizeof(tlsinstr_gd_le));
1045     return (FIX_RELOC);
1046
1047 #endif /* ! codereview */
1048
1049     ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_BADTLSINS),
1050                conv_reloc_amd64_type(arsp->rel_rtype, 0, &inv_buf),
1051                arsp->rel_isdesc->is_file->ifl_name,
1052                ld_reloc_sym_name(arsp),
1053                arsp->rel_isdesc->is_name,
1054                EC_OFF(arsp->rel_roffset));
1055     return (FIX_ERROR);
1056
1057 }
1058
1059 #endif /* ! codereview */
1060
1061     /* LD -> LE
1062
1063     * Transition
1064     *      0x00 leaq x1@tlsgd(%rip), %rdi
1065     *      0x07 call __tls_get_addr@plt
1066     *      0x0c
1067     * To:
1068     *      0x00 .byte 0x66
1069     *      0x01 .byte 0x66
1070     *      0x02 .byte 0x66
1071     *      0x03 movq %fs:0, %rax
1072
1073     */
1074     DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
1075                                   R_AMD64_NONE, arsp, ld_reloc_sym_name));
1076     offset -= 3;
1077     (void) memcpy(offset, tlsinstr_ld_le, sizeof(tlsinstr_ld_le));
1078     return (FIX_DONE);
1079
1080     /* LD->LE
1081
1082     * Transition:
1083     *      0x00 leaq x1@dtloff(%rax), %rcx
1084     * To:
1085     *      0x00 leaq x1@tpoff(%rax), %rcx
1086
1087     */
1088     DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
1089                                   R_AMD64_TPOFF32, arsp, ld_reloc_sym_name));
1090     arsp->rel_rtype = R_AMD64_TPOFF32;
1091     arsp->rel_raddend = 0;
1092     return (FIX_RELOC);
1093
1094
1095     static uintptr_t
1096     ld_do_activerelocs(Ofl_desc *ofl)
1097     {
1098         Rel_desc        *arsp;
1099         Rel_cachebuf   *rcbp;
1100
1101     }
1102
1103     /* Same code sequence used in the GD -> LE transition.
1104
1105     */
1106     if (offset[0] == REX_RW)
1107         offset[0] = REX_BW;
1108
1109     offset[1] = INSN_LEA;
1110     offset[2] = 0x80 | (reg << 3) | reg;
1111
1112     (void) memcpy(offset, tlsinstr_gd_le, sizeof(tlsinstr_gd_le));
1113     return (FIX_RELOC);
1114
1115 #endif /* ! codereview */
1116
1117     ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_BADTLSINS),
1118                conv_reloc_amd64_type(arsp->rel_rtype, 0, &inv_buf),
1119                arsp->rel_isdesc->is_file->ifl_name,
1120                ld_reloc_sym_name(arsp),
1121                arsp->rel_isdesc->is_name,
1122                EC_OFF(arsp->rel_roffset));
1123     return (FIX_ERROR);
1124
1125 }
1126
1127 #endif /* ! codereview */
1128
1129     /* LD -> LE
1130
1131     * Transition
1132     *      0x00 leaq x1@tlsgd(%rip), %rdi
1133     *      0x07 call __tls_get_addr@plt
1134     *      0x0c
1135     * To:
1136     *      0x00 .byte 0x66
1137     *      0x01 .byte 0x66
1138     *      0x02 .byte 0x66
1139     *      0x03 movq %fs:0, %rax
1140
1141     */
1142     DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
1143                                   R_AMD64_NONE, arsp, ld_reloc_sym_name));
1144     offset -= 3;
1145     (void) memcpy(offset, tlsinstr_ld_le, sizeof(tlsinstr_ld_le));
1146     return (FIX_DONE);
1147
1148     /* LD->LE
1149
1150     * Transition:
1151     *      0x00 leaq x1@dtloff(%rax), %rcx
1152     * To:
1153     *      0x00 leaq x1@tpoff(%rax), %rcx
1154
1155     */
1156     DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
1157                                   R_AMD64_TPOFF32, arsp, ld_reloc_sym_name));
1158     arsp->rel_rtype = R_AMD64_TPOFF32;
1159     arsp->rel_raddend = 0;
1160     return (FIX_RELOC);
1161
1162
1163     static uintptr_t
1164     ld_do_activerelocs(Ofl_desc *ofl)
1165     {
1166         Rel_desc        *arsp;
1167         Rel_cachebuf   *rcbp;
1168
1169     }
1170
1171     /* Same code sequence used in the GD -> LE transition.
1172
1173     */
1174     if (offset[0] == REX_RW)
1175         offset[0] = REX_BW;
1176
1177     offset[1] = INSN_LEA;
1178     offset[2] = 0x80 | (reg << 3) | reg;
1179
1180     (void) memcpy(offset, tlsinstr_gd_le, sizeof(tlsinstr_gd_le));
1181     return (FIX_RELOC);
1182
1183 #endif /* ! codereview */
1184
1185     ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_BADTLSINS),
1186                conv_reloc_amd64_type(arsp->rel_rtype, 0, &inv_buf),
1187                arsp->rel_isdesc->is_file->ifl_name,
1188                ld_reloc_sym_name(arsp),
1189                arsp->rel_isdesc->is_name,
1190                EC_OFF(arsp->rel_roffset));
1191     return (FIX_ERROR);
1192
1193 }
1194
1195 #endif /* ! codereview */
1196
1197     /* LD -> LE
1198
1199     * Transition
1200     *      0x00 leaq x1@tlsgd(%rip), %rdi
1201     *      0x07 call __tls_get_addr@plt
1202     *      0x0c
1203     * To:
1204     *      0x00 .byte 0x66
1205     *      0x01 .byte 0x66
1206     *      0x02 .byte 0x66
1207     *      0x03 movq %fs:0, %rax
1208
1209     */
1210     DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
1211                                   R_AMD64_NONE, arsp, ld_reloc_sym_name));
1212     offset -= 3;
1213     (void) memcpy(offset, tlsinstr_ld_le, sizeof(tlsinstr_ld_le));
1214     return (FIX_DONE);
1215
1216     /* LD->LE
1217
1218     * Transition:
1219     *      0x00 leaq x1@dtloff(%rax), %rcx
1220     * To:
1221     *      0x00 leaq x1@tpoff(%rax), %rcx
1222
1223     */
1224     DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
1225                                   R_AMD64_TPOFF32, arsp, ld_reloc_sym_name));
1226     arsp->rel_rtype = R_AMD64_TPOFF32;
1227     arsp->rel_raddend = 0;
1228     return (FIX_RELOC);
1229
1230
1231     static uintptr_t
1232     ld_do_activerelocs(Ofl_desc *ofl)
1233     {
1234         Rel_desc        *arsp;
1235         Rel_cachebuf   *rcbp;
1236
1237     }
1238
1239     /* Same code sequence used in the GD -> LE transition.
1240
1241     */
1242     if (offset[0] == REX_RW)
1243         offset[0] = REX_BW;
1244
1245     offset[1] = INSN_LEA;
1246     offset[2] = 0x80 | (reg << 3) | reg;
1247
1248     (void) memcpy(offset, tlsinstr_gd_le, sizeof(tlsinstr_gd_le));
1249     return (FIX_RELOC);
1250
1251 #endif /* ! codereview */
1252
1253     ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_BADTLSINS),
1254                conv_reloc_amd64_type(arsp->rel_rtype, 0, &inv_buf),
1255                arsp->rel_isdesc->is_file->ifl_name,
1256                ld_reloc_sym_name(arsp),
1257                arsp->rel_isdesc->is_name,
1258                EC_OFF(arsp->rel_roffset));
1259     return (FIX_ERROR);
1260
1261 }
1262
1263 #endif /* ! codereview */
1264
1265     /* LD -> LE
1266
1267     * Transition
1268     *      0x00 leaq x1@tlsgd(%rip), %rdi
1269     *      0x07 call __tls_get_addr@plt
1270     *      0x0c
1271     * To:
1272     *      0x00 .byte 0x66
1273     *      0x01 .byte 0x66
1274     *      0x02 .byte 0x66
1275     *      0x03 movq %fs:0, %rax
1276
1277     */
1278     DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
1279                                   R_AMD64_NONE, arsp, ld_reloc_sym_name));
1280     offset -= 3;
1281     (void) memcpy(offset, tlsinstr_ld_le, sizeof(tlsinstr_ld_le));
1282     return (FIX_DONE);
1283
1284     /* LD->LE
1285
1286     * Transition:
1287     *      0x00 leaq x1@dtloff(%rax), %rcx
1288     * To:
1289     *      0x00 leaq x1@tpoff(%rax), %rcx
1290
1291     */
1292     DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
1293                                   R_AMD64_TPOFF32, arsp, ld_reloc_sym_name));
1294     arsp->rel_rtype = R_AMD64_TPOFF32;
1295     arsp->rel_raddend = 0;
1296     return (FIX_RELOC);
1297
1298
1299     static uintptr_t
1300     ld_do_activerelocs(Ofl_desc *ofl)
1301     {
1302         Rel_desc        *arsp;
1303         Rel_cachebuf   *rcbp;
1304
1305     }
1306
1307     /* Same code sequence used in the GD -> LE transition.
1308
1309     */
1310     if (offset[0] == REX_RW)
1311         offset[0] = REX_BW;
1312
1313     offset[1] = INSN_LEA;
1314     offset[2] = 0x80 | (reg << 3) | reg;
1315
1316     (void) memcpy(offset, tlsinstr_gd_le, sizeof(tlsinstr_gd_le));
1317     return (FIX_RELOC);
1318
1319 #endif /* ! codereview */
1320
1321     ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_BADTLSINS),
1322                conv_reloc_amd64_type(arsp->rel_rtype, 0, &inv_buf),
1323                arsp->rel_isdesc->is_file->ifl_name,
1324                ld_reloc_sym_name(arsp),
1325                arsp->rel_isdesc->is_name,
1326                EC_OFF(arsp->rel_roffset));
1327     return (FIX_ERROR);
1328
1329 }
1330
1331 #endif /* ! codereview */
1332
1333     /* LD -> LE
1334
1335     * Transition
1336     *      0x00 leaq x1@tlsgd(%rip), %rdi
1337     *      0x07 call __tls_get_addr@plt
1338     *      0x0c
1339     * To:
1340     *      0x00 .byte 0x66
1341     *      0x01 .byte 0x66
1342     *      0x02 .byte 0x66
1343     *      0x03 movq %fs:0, %rax
1344
1345     */
1346     DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
1347                                   R_AMD64_NONE, arsp, ld_reloc_sym_name));
1348     offset -= 3;
1349     (void) memcpy(offset, tlsinstr_ld_le, sizeof(tlsinstr_ld_le));
1350     return (FIX_DONE);
1351
1352     /* LD->LE
1353
1354     * Transition:
1355     *      0x00 leaq x1@dtloff(%rax), %rcx
1356     * To:
1357     *      0x00 leaq x1@tpoff(%rax), %rcx
1358
1359     */
1360     DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
1361                                   R_AMD64_TPOFF32, arsp, ld_reloc_sym_name));
1362     arsp->rel_rtype = R_AMD64_TPOFF32;
1363     arsp->rel_raddend = 0;
1364     return (FIX_RELOC);
1365
1366
1367     static uintptr_t
1368     ld_do_activerelocs(Ofl_desc *ofl)
1369     {
1370         Rel_desc        *arsp;
1371         Rel_cachebuf   *rcbp;
1372
1373     }
1374
1375     /* Same code sequence used in the GD -> LE transition.
1376
1377     */
1378     if (offset[0] == REX_RW)
1379         offset[0] = REX_BW;
1380
1381     offset[1] = INSN_LEA;
1382     offset[2] = 0x80 | (reg << 3) | reg;
1383
1384     (void) memcpy(offset, tlsinstr_gd_le, sizeof(tlsinstr_gd_le));
1385     return (FIX_RELOC);
1386
1387 #endif /* ! codereview */
1388
1389     ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_BADTLSINS),
1390                conv_reloc_amd64_type(arsp->rel_rtype, 0, &inv_buf),
1391                arsp->rel_isdesc->is_file->ifl_name,
1392                ld_reloc_sym_name(arsp),
1393                arsp->rel_isdesc->is_name,
1394                EC_OFF(arsp->rel_roffset));
1395     return (FIX_ERROR);
1396
1397 }
1398
1399 #endif /* ! codereview */
1400
1401     /* LD -> LE
1402
1403     * Transition
1404     *      0x00 leaq x1@tlsgd(%rip), %rdi
1405     *      0x07 call __tls_get_addr@plt
1406     *      0x0c
1407     * To:
1408     *      0x00 .byte 0x66
1409     *      0x01 .byte 0x66
1410     *      0x02 .byte 0x66
1411     *      0x03 movq %fs:0, %rax
1412
1413     */
1414     DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
1415                                   R_AMD64_NONE, arsp, ld_reloc_sym_name));
1416     offset -= 3;
1417     (void) memcpy(offset, tlsinstr_ld_le, sizeof(tlsinstr_ld_le));
1418     return (FIX_DONE);
1419
1420     /* LD->LE
1421
1422     * Transition:
1423     *      0x00 leaq x1@dtloff(%rax), %rcx
1424     * To:
1425     *      0x00 leaq x1@tpoff(%rax), %rcx
1426
1427     */
1428     DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
1429                                   R_AMD64_TPOFF32, arsp, ld_reloc_sym_name));
1430     arsp->rel_rtype = R_AMD64_TPOFF32;
1431     arsp->rel_raddend = 0;
1432     return (FIX_RELOC);
1433
1434
1435     static uintptr_t
1436     ld_do_activerelocs(Ofl_desc *ofl)
1437     {
1438         Rel_desc        *arsp;
1439         Rel_cachebuf   *rcbp;
1440
1441     }
1442
1443     /* Same code sequence used in the GD -> LE transition.
1444
1445     */
1446     if (offset[0] == REX_RW)
1447         offset[0] = REX_BW;
1448
1449     offset[1] = INSN_LEA;
1450     offset[2] = 0x80 | (reg << 3) | reg;
1451
1452     (void) memcpy(offset, tlsinstr_gd_le, sizeof(tlsinstr_gd_le));
1453     return (FIX_RELOC);
1454
1455 #endif /* ! codereview */
1456
1457     ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_BADTLSINS),
1458                conv_reloc_amd64_type(arsp->rel_rtype, 0, &inv_buf),
1459                arsp->rel_isdesc->is_file->ifl_name,
1460                ld_reloc_sym_name(arsp),
1461                arsp->rel_isdesc->is_name,
1462                EC_OFF(arsp->rel_roffset));
1463     return (FIX_ERROR);
1464
1465 }
1466
1467 #endif /* ! codereview */
1468
1469     /* LD -> LE
1470
1471     * Transition
1472     *      0x00 leaq x1@tlsgd(%rip), %rdi
1473     *      0x07 call __tls_get_addr@plt
1474     *      0x0c
1475     * To:
1476     *      0x00 .byte 0x66
1477     *      0x01 .byte 0x66
1478     *      0x02 .byte 0x66
1479     *      0x03 movq %fs:0, %rax
1480
1481     */
1482     DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
1483                                   R_AMD64_NONE, arsp, ld_reloc_sym_name));
1484     offset -= 3;
1485     (void) memcpy(offset, tlsinstr_ld_le, sizeof(tlsinstr_ld_le));
1486     return (FIX_DONE);
1487
1488     /* LD->LE
1489
1490     * Transition:
1491     *      0x00 leaq x1@dtloff(%rax), %rcx
1492     * To:
1493     *      0x00 leaq x1@tpoff(%rax), %rcx
1494
1495     */
1496     DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
1497                                   R_AMD64_TPOFF32, arsp, ld_reloc_sym_name));
1498     arsp->rel_rtype = R_AMD64_TPOFF32;
1499     arsp->rel_raddend = 0;
1500     return (FIX_RELOC);
1501
1502
1503     static uintptr_t
1504     ld_do_activerelocs(Ofl_desc *ofl)
1505     {
1506         Rel_desc        *arsp;
1507         Rel_cachebuf   *rcbp;
1508
1509     }
1510
1511     /* Same code sequence used in
```

```

761     Aliste          idx;
762     uintptr_t      return_code = 1;
763     ofl_flag_t     flags = ofl->ofl_flags;
764
765     if (aplist_nitems(ofl->ofl_actrels.rc_list) != 0)
766         DBG_CALL(Dbg_reloc_doact_title(ofl->ofl_lml));
767
768     /*
769      * Process active relocations.
770     */
771     REL_CACHE_TRAVERSE(&ofl->ofl_actrels, idx, rcbp, arsp) {
772         uchar_t          *addr;
773         Xword            value;
774         Sym_desc         *sdp;
775         const char       *ifl_name;
776         Xword            refaddr;
777         int               moved = 0;
778         Gotref           gref;
779         Os_desc          *osp;
780
781         /*
782          * If the section this relocation is against has been discarded
783          * (-zignore), then discard (skip) the relocation itself.
784         */
785         if ((arsp->rel_isdesc->is_flags & FLG_IS_DISCARD) &&
786             ((arsp->rel_flags & (FLG_REL_GOT | FLG_REL_BSS |
787                 FLG_REL_PLT | FLG_REL_NOINFO)) == 0)) {
788             DBG_CALL(Dbg_reloc_discard(ofl->ofl_lml, M_MACH, arsp));
789             continue;
790         }
791
792         /*
793          * We determine what the 'got reference' model (if required)
794          * is at this point. This needs to be done before tls_fixup()
795          * since it may 'transition' our instructions.
796         */
797
798         * The got table entries have already been assigned,
799         * and we bind to those initial entries.
800         */
801         if (arsp->rel_flags & FLG_REL_DTLS)
802             gref = GOT_REF_TLSGD;
803         else if (arsp->rel_flags & FLG_REL_MTLS)
804             gref = GOT_REF_TLSDL;
805         else if (arsp->rel_flags & FLG_REL_STLS)
806             gref = GOT_REF_TLSIE;
807         else
808             gref = GOT_REF_GENERIC;
809
810         /*
811          * Perform any required TLS fixups.
812         */
813         if (arsp->rel_flags & FLG_REL_TLSFIX) {
814             Fixupret        ret;
815
816             if ((ret = tls_fixups(ofl, arsp)) == FIX_ERROR)
817                 return (S_ERROR);
818             if (ret == FIX_DONE)
819                 continue;
820
821             /*
822              * If this is a relocation against a move table, or
823              * expanded move table, adjust the relocation entries.
824             */
825             if (RELAUX_GET_MOVE(arsp))
826                 ld_adj_movereloc(ofl, arsp);

```

```

828     sdp = arsp->rel_sym;
829     refaddr = arsp->rel_roffset +
830             (Off)_elf_getxoff(arsp->rel_isdesc->is_indata);
831
832     if ((arsp->rel_flags & FLG_REL_CLVAL) ||
833         (arsp->rel_flags & FLG_REL_GOTCL))
834         value = 0;
835     else if (ELF_ST_TYPE(sdp->sd_sym->st_info) == STT_SECTION) {
836         Sym_desc         *sym;
837
838         /*
839          * The value for a symbol pointing to a SECTION
840          * is based off of that sections position.
841         */
842         if ((sdp->sd_isc->is_flags & FLG_IS_RELUPD) &&
843             /* LINTED */
844             (sym = ld_am_I_partial(arsp, arsp->rel_raddend))) {
845             /*
846              * The symbol was moved, so adjust the value
847              * relative to the new section.
848             */
849             value = sym->sd_sym->st_value;
850             moved = 1;
851
852             /*
853              * The original raddend covers the displacement
854              * from the section start to the desired
855              * address. The value computed above gets us
856              * from the section start to the start of the
857              * symbol range. Adjust the old raddend to
858              * remove the offset from section start to
859              * symbol start, leaving the displacement
860              * within the range of the symbol.
861             */
862             arsp->rel_raddend -= sym->sd_osym->st_value;
863         } else {
864             value = _elf_getxoff(sdp->sd_isc->is_indata);
865             if (sdp->sd_isc->is_shdr->sh_flags & SHF_ALLOC)
866                 value += sdp->sd_isc->is_osdesc->
867                         os_shdr->sh_addr;
868             if (sdp->sd_isc->is_shdr->sh_flags & SHF_TLS)
869                 value -= ofl->ofl_tlsphdr->p_vaddr;
870
871             /*
872              * If the symbol is a TLS symbol, then we
873              * need to add the TLS offset to the value.
874             */
875             if (IS_SIZE(arsp->rel_rtype)) {
876                 /*
877                  * Size relocations require the symbols size.
878                  */
879                 value = sdp->sd_sym->st_size;
880
881             } else if (((sdp->sd_flags & FLG_SY_CAP) &&
882                         sdp->sd_aux && sdp->sd_aux->sa_PLTndx) {
883                 /*
884                  * If relocation is against a capabilities symbol, we
885                  * need to jump to an associated PLT, so that at runtime
886                  * ld.so.1 is involved to determine the best binding
887                  * choice. Otherwise, the value is the symbols value.
888                 */
889                 value = ld_calc_plt_addr(sdp, ofl);
890             } else
891                 value = sdp->sd_sym->st_value;
892
893             /*
894              * Relocation against the GLOBAL_OFFSET_TABLE.
895              */
896         }

```

```

893     if ((arsp->rel_flags & FLG_REL_GOT) &&
894         !ld_reloc_set_aux_osdesc(ofl, arsp, ofl->ofl_ogot))
895         return (S_ERROR);
896     osp = RELAUX_GET_OSDESC(arsp);

898 /*
899 * If loadable and not producing a relocatable object add the
900 * sections virtual address to the reference address.
901 */
902 if ((arsp->rel_flags & FLG_REL_LOAD) &&
903     ((flags & FLG_OF_RELOBJ) == 0))
904     refaddr += arsp->rel_isdesc->is_osdesc->
905                 os_shdr->sh_addr;

907 /*
908 * If this entry has a PLT assigned to it, its value is actually
909 * the address of the PLT (and not the address of the function).
910 */
911 if (IS_PLT(arsp->rel_rtype)) {
912     if (sdp->sd_aux && sdp->sd_aux->sa_PLTndx)
913         value = ld_calc_plt_addr(sdp, ofl);
914 }

916 /*
917 * Add relocations addend to value. Add extra
918 * relocation addend if needed.
919 */
920 /*
921 * Note: For GOT relative relocations on amd64 we discard the
922 * addend. It was relevant to the reference - not to the
923 * data item being referenced (ie: that -4 thing).
924 */
925 if ((arsp->rel_flags & FLG_REL_GOT) == 0)
926     value += arsp->rel_raddend;

927 /*
928 * Determine whether the value needs further adjustment. Filter
929 * through the attributes of the relocation to determine what
930 * adjustment is required. Note, many of the following cases
931 * are only applicable when a .got is present. As a .got is
932 * not generated when a relocatable object is being built,
933 * any adjustments that require a .got need to be skipped.
934 */
935 if ((arsp->rel_flags & FLG_REL_GOT) &&
936     ((flags & FLG_OF_RELOBJ) == 0)) {
937     Xword             Rladdr;
938     uintptr_t          R2addr;
939     Word              gotndx;
940     Gotndx           *gnp;

942 /*
943 * Perform relocation against GOT table. Since this
944 * doesn't fit exactly into a relocation we place the
945 * appropriate byte in the GOT directly
946 */
947 /*
948 * Calculate offset into GOT at which to apply
949 * the relocation.
950 */
951 gnp = ld_find_get_ndx(sdp->sd_GOTndxs, gref, ofl, arsp);
952 assert(gnp);

953 if (arsp->rel_rtype == R_AMD64_DTPOFF64)
954     gotndx = gnp->gn_gotndx + 1;
955 else
956     gotndx = gnp->gn_gotndx;

958 Rladdr = (Xword)(gotndx * M_GOT_ENTSIZE);

```

```

960 /*
961 * Add the GOTs data's offset.
962 */
963 R2addr = Rladdr + (uintptr_t)osp->os_outdata->d_buf;

965 DBG_CALL(Dbg_reloc_doact(ofl->ofl_lml, ELF_DBG_LD_ACT,
966                           M_MACH, SHT_REL, arsp, Rladdr, value,
967                           ld_reloc_sym_name));

969 /*
970 * And do it.
971 */
972 if (ofl->ofl_flags1 & FLG_OF1_ENCDIFF)
973     *(Xword *)R2addr = ld_bswap_Xword(value);
974 else
975     *(Xword *)R2addr = value;
976 continue;

978 } else if (IS_GOT_BASED(arsp->rel_rtype) &&
979             ((flags & FLG_OF_RELOBJ) == 0)) {
980     value -= ofl->ofl_ogot->os_shdr->sh_addr;

982 } else if (IS_GOTPCREL(arsp->rel_rtype) &&
983             ((flags & FLG_OF_RELOBJ) == 0)) {
984     Gotndx *gnp;

986 /*
987 * Calculation:
988 *   G + GOT + A - P
989 */
990 gnp = ld_find_get_ndx(sdp->sd_GOTndxs, gref, ofl, arsp);
991 assert(gnp);
992 value = (Xword)(ofl->ofl_ogot->os_shdr->sh_addr) +
993         ((Xword)gnp->gn_gotndx * M_GOT_ENTSIZE) +
994         arsp->rel_raddend - refaddr;

996 } else if (IS_GOT_PC(arsp->rel_rtype) &&
997             ((flags & FLG_OF_RELOBJ) == 0)) {
998     value = (Xword)(ofl->ofl_ogot->os_shdr->
999                     sh_addr) - refaddr + arsp->rel_raddend;

1001 } else if ((IS_PC_RELATIVE(arsp->rel_rtype) &&
1002             (((flags & FLG_OF_RELOBJ) == 0) ||
1003             (osp == sdp->sd_isc->is_osdesc))) {
1004     value -= refaddr;

1006 } else if (IS_TLS_INS(arsp->rel_rtype) &&
1007             IS_GOT_RELATIVE(arsp->rel_rtype) &&
1008             ((flags & FLG_OF_RELOBJ) == 0)) {
1009     Gotndx *gnp;

1011 gnp = ld_find_get_ndx(sdp->sd_GOTndxs, gref, ofl, arsp);
1012 assert(gnp);
1013 value = (Xword)gnp->gn_gotndx * M_GOT_ENTSIZE;

1015 } else if (IS_GOT_RELATIVE(arsp->rel_rtype) &&
1016             ((flags & FLG_OF_RELOBJ) == 0)) {
1017     Gotndx *gnp;

1019 gnp = ld_find_get_ndx(sdp->sd_GOTndxs, gref, ofl, arsp);
1020 assert(gnp);
1021 value = (Xword)gnp->gn_gotndx * M_GOT_ENTSIZE;

1023 } else if ((arsp->rel_flags & FLG_REL_STLS) &&
1024             ((flags & FLG_OF_RELOBJ) == 0)) {

```

new/usr/src/cmd/sgs/libld/common/machrel.amd.c

19

```

1025           Xword    tlsstatsize;
1027
1028           /*
1029           * This is the LE TLS reference model.  Static
1030           * offset is hard-coded.
1031           */
1032           tlsstatsize = S_ROUND(ofl->ofl_tlsphdr->p_memsz,
1033                               M_TLSSTATALIGN);
1034           value = tlsstatsize - value;
1035
1036           /*
1037           * Since this code is fixed up, it assumes a negative
1038           * offset that can be added to the thread pointer.
1039           */
1040           if (arsp->rel_rtype == R_AMD64_TPOFF32)
1041               value = -value;
1042
1043       if (arsp->rel_isdesc->is_file)
1044           ifl_name = arsp->rel_isdesc->is_file->ifl_name;
1045       else
1046           ifl_name = MSG_INTL(MSG_STR_NULL);
1047
1048       /*
1049       * Make sure we have data to relocate. Compiler and assembler
1050       * developers have been known to generate relocations against
1051       * invalid sections (normally .bss), so for their benefit give
1052       * them sufficient information to help analyze the problem.
1053       * End users should never see this.
1054       */
1055       if (arsp->rel_isdesc->is_indata->d_buf == 0) {
1056           Conv_inv_buf_t inv_buf;
1057
1058           ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_EMPTYSEC),
1059                      conv_reloc_amd64_type(arsp->rel_rtype, 0, &inv_buf),
1060                      ifl_name, ld_reloc_sym_name(arsp),
1061                      EC_WORD(arsp->rel_isdesc->is_scnndx),
1062                      arsp->rel_isdesc->is_name);
1063           return (S_ERROR);
1064       }
1065
1066       /*
1067       * Get the address of the data item we need to modify.
1068       */
1069       addr = ((uchar_t *)((uintptr_t)arsp->rel_roffset +
1070                           (uintptr_t)_elf_getxoff(arsp->rel_isdesc->is_indata)));
1071
1072       DBG_CALL(Dbg_reloc_doact(ofl->ofl_lml, ELF_DBG_LD_ACT,
1073                               M_MACH, SHT_REL_A, arsp, EC_NATPTR(addr), value,
1074                               ld_reloc_sym_name));
1075       addr += ((uintptr_t)osp->os_outdata->d_buf);
1076
1077       if (((uintptr_t)addr - (uintptr_t)ofl->ofl_nehdr) >
1078           (ofl->ofl_size) || (arsp->rel_roffset >
1079                               osp->os_shdr->sh_size)) {
1080           int             class;
1081           Conv_inv_buf_t inv_buf;
1082
1083           if (((uintptr_t)addr - (uintptr_t)ofl->ofl_nehdr) >
1084               (ofl->ofl_size))
1085               class = ERR_FATAL;
1086           else
1087               class = ERR_WARNING;
1088
1089           ld_eprintf(ofl, class, MSG_INTL(MSG_REL_INVALOFFSET),
1090                      conv_reloc_amd64_type(arsp->rel_rtype, 0, &inv_buf));

```

new/usr/src/cmd/sgs/libld/common/machrel.amd.c

```

1091 ifl_name, EC_WORD(arsp->rel_isdesc->is_scnndx),
1092 arsp->rel_isdesc->is_name, ld_reloc_sym_name(arsp),
1093 EC_ADDR((uintptr_t)addr -
1094 (uintptr_t)ofl->ofl_nehdr));
1095
1096 if (class == ERR_FATAL) {
1097     return_code = S_ERROR;
1098     continue;
1099 }
1100 }
1101
1102 /*
1103 * The relocation is additive. Ignore the previous symbol
1104 * value if this local partial symbol is expanded.
1105 */
1106 if (moved)
1107     value -= *addr;
1108
1109 /*
1110 * If '-z noreloc' is specified - skip the do_reloc_ld stage.
1111 */
1112 if (OFL_DO_RELOC(ofl)) {
1113     /*
1114     * If this is a PROGBITS section and the running linker
1115     * has a different byte order than the target host,
1116     * tell do_reloc_ld() to swap bytes.
1117     */
1118     if (do_reloc_ld(arsp, addr, &value, ld_reloc_sym_name,
1119                     ifl_name, OFL_SWAP_RELLOC_DATA(ofl, arsp),
1120                     ofl->ofl_lml) == 0) {
1121         ofl->ofl_flags |= FLG_OF_FATAL;
1122         return_code = S_ERROR;
1123     }
1124 }
1125
1126 return (return_code);
1127 }

1128 static uintptr_t
1129 ld_add_outrel(Word flags, Rel_desc *rsp, Ofl_desc *ofl)
1130 {
1131     Rel_desc      *orssp;
1132     Sym_desc      *sdp = rsp->rel_sym;
1133
1134     /*
1135     * Static executables *do not* want any relocations against them.
1136     * Since our engine still creates relocations against a WEAK UNDEFINED
1137     * symbol in a static executable, it's best to disable them here
1138     * instead of through out the relocation code.
1139     */
1140     if (OFL_IS_STATIC_EXEC(ofl))
1141         return (1);
1142
1143     /*
1144     * If we are adding a output relocation against a section
1145     * symbol (non-RELATIVE) then mark that section. These sections
1146     * will be added to the .dynsym symbol table.
1147     */
1148     if (sdp && (rsp->rel_rtype != M_R_RELATIVE) &&
1149         ((flags & FLG_REL_SCNNDX) ||
1150          (ELF_ST_TYPE(sdp->sd_sym->st_info) == STT_SECTION))) {
1151
1152     /*
1153     * If this is a COMMON symbol - no output section
1154     * exists yet - (it's created as part of sym_validate()).
1155     * So - we mark here that when it's created it should
1156

```

```

1157         * be tagged with the FLG_OS_OUTREL flag.
1158         */
1159     if ((sdp->sd_flags & FLG_SY_SPECSEC) &&
1160         (sdp->sd_sym->st_shndx == SHN_COMMON)) {
1161         if (ELF_ST_TYPE(sdp->sd_sym->st_info) != STT_TLS)
1162             ofl->ofl_flags1 |= FLG_OF1_BSSOREL;
1163         else
1164             ofl->ofl_flags1 |= FLG_OF1_TLSOREL;
1165     } else {
1166         Os_desc *osp;
1167         Is_desc *isp = sdp->sd_isc;
1168
1169         if (isp && ((osp = isp->is_osdesc) != NULL) &&
1170             ((osp->os_flags & FLG_OS_OUTREL) == 0)) {
1171             ofl->ofl_dynshdrcnt++;
1172             osp->os_flags |= FLG_OS_OUTREL;
1173         }
1174     }
1175
1176     /* Enter it into the output relocation cache */
1177     if ((orssp = ld_reloc_enter(ofl, &ofl->ofl_outrels, rsp, flags)) == NULL)
1178         return (S_ERROR);
1179
1180     if (flags & FLG_REL_GOT)
1181         ofl->ofl_relocgotsz += (Xword)sizeof (Rela);
1182     else if (flags & FLG_REL_PLT)
1183         ofl->ofl_relocpltsz += (Xword)sizeof (Rela);
1184     else if (flags & FLG_REL_BSS)
1185         ofl->ofl_relocbsssz += (Xword)sizeof (Rela);
1186     else if (flags & FLG_REL_NOINFO)
1187         ofl->ofl_relocrelsz += (Xword)sizeof (Rela);
1188     else
1189         RELAUX_GET_OSDESC(orssp)->os_szoutrels += (Xword)sizeof (Rela);
1190
1191     if (orssp->rel_rtype == M_R_RELATIVE)
1192         ofl->ofl_relocrelcnt++;
1193
1194     /*
1195      * We don't perform sorting on PLT relocations because
1196      * they have already been assigned a PLT index and if we
1197      * were to sort them we would have to re-assign the plt indexes.
1198      */
1199     if (!(flags & FLG_REL_PLT))
1200         ofl->ofl_reloccnt++;
1201
1202     /*
1203      * Insure a GLOBAL_OFFSET_TABLE is generated if required.
1204      */
1205     if (IS_GOT_REQUIRED(orssp->rel_rtype))
1206         ofl->ofl_flags |= FLG_OF_BLDGOT;
1207
1208     /*
1209      * Identify and possibly warn of a displacement relocation.
1210      */
1211     if (orssp->rel_flags & FLG_REL_DISP) {
1212         ofl->ofl_dtflags_1 |= DF_1_DISPRELPND;
1213
1214         if (ofl->ofl_flags & FLG_OF_VERBOSE)
1215             ld_disp_errmsg(MSG_INTL(MSG_REL_DISPREL4), orssp, ofl);
1216     }
1217     DBG_CALL(Debug_reloc_ors_entry(ofl->ofl_lml, ELF_DBG_LD, SHT_REL,
1218                                     M_MACH, orssp));
1219
1220     return (1);
1221 }
```

```

1223 /*
1224  * process relocation for a LOCAL symbol
1225 */
1226 static uintptr_t
1227 ld_reloc_local(Rel_desc * rsp, Ofl_desc * ofl)
1228 {
1229     ofl_flag_t    flags = ofl->ofl_flags;
1230     Sym_desc     *sdp = rsp->rel_sym;
1231     Word          shndx = sdp->sd_sym->st_shndx;
1232     Word          ortype = rsp->rel_rtype;
1233
1234     /*
1235      * if ((shared object) and (not pc relative relocation) and
1236      *      (not against ABS symbol))
1237      * then
1238      *      build R_AMD64_RELATIVE
1239      * fi
1240     */
1241     if ((flags & FLG_OF_SHAROBJ) && (rsp->rel_flags & FLG_REL_LOAD) &&
1242         !(IS_PC_RELATIVE(rsp->rel_rtype)) && !(IS_SIZE(rsp->rel_rtype)) &&
1243         !(IS_GOT_BASED(rsp->rel_rtype)) &&
1244         !(rsp->rel_isdesc != NULL) &&
1245         (rsp->rel_isdesc->is_shdr->sh_type == SHT_SUNW_dof) &&
1246         (((sdp->sd_flags & FLG_SY_SPECSEC) == 0) ||
1247          (shndx != SHN_ABS) || (sdp->sd_aux && sdp->sd_aux->sa_symspec))) {
1248
1249     /*
1250      * R_AMD64_RELATIVE updates a 64bit address, if this
1251      * relocation isn't a 64bit binding then we can not
1252      * simplify it to a RELATIVE relocation.
1253     */
1254     if (reloc_table[ortype].re_fsize != sizeof (Addr)) {
1255         return (ld_add_outrel(0, rsp, ofl));
1256     }
1257
1258     rsp->rel_rtype = R_AMD64_RELATIVE;
1259     if (ld_add_outrel(FLG_REL_ADVVAL, rsp, ofl) == S_ERROR)
1260         return (S_ERROR);
1261     rsp->rel_rtype = ortype;
1262
1263 }
1264
1265 /*
1266  * If the relocation is against a 'non-allocatable' section
1267  * and we can not resolve it now - then give a warning
1268  * message.
1269 */
1270
1271 /*
1272  * We can not resolve the symbol if either:
1273  *   a) it's undefined
1274  *   b) it's defined in a shared library and a
1275  *      COPY relocation hasn't moved it to the executable
1276 */
1277
1278 /*
1279  * Note: because we process all of the relocations against the
1280  * text segment before any others - we know whether
1281  * or not a copy relocation will be generated before
1282  * we get here (see reloc_init()->reloc_segments()).
1283 */
1284 if (!(rsp->rel_flags & FLG_REL_LOAD) &&
1285     ((shndx == SHN_UNDEF) ||
1286      ((sdp->sd_ref == REF_DYN_NEED) &&
1287       ((sdp->sd_flags & FLG_SY_MVTOCOMM) == 0)))) {
1288     Conv_inv_buf_t inv_buf;
1289     Os_desc        *osp = RELAUX_GET_OSDESC(rsp);
1290
1291     /*
1292      * If the relocation is against a SHT_SUNW_ANNOTATE
1293     */
1294 }
```

```

1289         * section - then silently ignore that the relocation
1290         * can not be resolved.
1291         */
1292     if (osp && (osp->os_shdr->sh_type == SHT_SUNW_ANNOTATE))
1293         return (0);
1294     ld_eprintf(ofl, ERR_WARNING, MSG_INTL(MSG_REL_EXTERNSYM),
1295                 conv_reloc_amd64_type(rsp->rel_rtype, 0, &inv_buf),
1296                 rsp->rel_isdesc->is_file->ifl_name,
1297                 ld_reloc_sym_name(rsp), osp->os_name);
1298     return (1);
1299 }
1300 /*
1301  * Perform relocation.
1302  */
1303 return (ld_add_actrel(NULL, rsp, ofl));
1305 }

1308 static uintptr_t
1309 ld_reloc_TLS(Boolean local, Rel_desc * rsp, Ofl_desc * ofl)
1310 {
1311     Word          rtype = rsp->rel_rtype;
1312     Sym_desc      *sdp = rsp->rel_sym;
1313     ofl_flag_t    flags = ofl->ofl_flags;
1314     Gotndx       *gnp;

1316 /*
1317  * If we're building an executable - use either the IE or LE access
1318  * model. If we're building a shared object process any IE model.
1319  */
1320 if ((flags & FLG_OF_EXEC) || (IS_TLS_IE(rtype))) {
1321     /*
1322      * Set the DF_STATIC_TLS flag.
1323      */
1324     ofl->ofl_dtflags |= DF_STATIC_TLS;

1326 if (!local || ((flags & FLG_OF_EXEC) == 0)) {
1327     /*
1328      * Assign a GOT entry for static TLS references.
1329      */
1330     if ((gnp = ld_find_got_ndx(sdp->sd_GOTndxs,
1331                               GOT_REF_TLSIE, ofl, rsp)) == NULL) {
1332
1333         if (ld_assign_got_TLS(local, rsp, ofl, sdp,
1334                               gnp, GOT_REF_TLSIE, FLG_REL_STLS,
1335                               rtype, R_AMD64_TPOFF64, 0) == S_ERROR)
1336             return (S_ERROR);
1337     }
1338
1339     /*
1340      * IE access model.
1341      */
1342     if (IS_TLS_IE(rtype))
1343         return (ld_add_actrel(FLG_REL_STLS, rsp, ofl));

1345     /*
1346      * Fixups are required for other executable models.
1347      */
1348     return (ld_add_actrel((FLG_REL_TLSFIX | FLG_REL_STLS),
1349                           rsp, ofl));
1350 }
1352 /*
1353  * LE access model.
1354  */

```

```

1355         if (IS_TLS_LE(rtype))
1356             return (ld_add_actrel(FLG_REL_STLS, rsp, ofl));
1358
1359         return (ld_add_actrel((FLG_REL_TLSFIX | FLG_REL_STLS),
1360                               rsp, ofl));
1360     }

1362 /*
1363  * Building a shared object.
1364  *
1365  * Assign a GOT entry for a dynamic TLS reference.
1366  */
1367 if (IS_TLS_LD(rtype) && ((gnp = ld_find_got_ndx(sdp->sd_GOTndxs,
1368                               GOT_REF_TLSLD, ofl, rsp)) == NULL)) {
1369
1370     if (ld_assign_got_TLS(local, rsp, ofl, sdp, gnp, GOT_REF_TLSLD,
1371                           FLG_REL_MTLS, rtype, R_AMD64_DTPMOD64, NULL) == S_ERROR)
1372         return (S_ERROR);

1374 } else if (IS_TLS_GD(rtype) &&
1375             ((gnp = ld_find_got_ndx(sdp->sd_GOTndxs, GOT_REF_TLSD,
1376                                     ofl, rsp)) == NULL)) {
1377
1378     if (ld_assign_got_TLS(local, rsp, ofl, sdp, gnp, GOT_REF_TLSD,
1379                           FLG_REL_DTLS, rtype, R_AMD64_DTPMOD64,
1380                           R_AMD64_DTPOFF64) == S_ERROR)
1381         return (S_ERROR);
1382     }

1384 if (IS_TLS_LD(rtype))
1385     return (ld_add_actrel(FLG_REL_MTLS, rsp, ofl));
1387
1388 return (ld_add_actrel(FLG_REL_DTLS, rsp, ofl));

1390 /* ARGUSED5 */
1391 static uintptr_t
1392 ld_assign_got_ndx(Alist **alpp, Gotndx *pgnp, Gotref gref, Ofl_desc *ofl,
1393                     Rel_desc *rsp, Sym_desc *sdp)
1394 {
1395     Xword          raddend;
1396     Gotndx       gn, *gnp;
1397     Aliste        idx;
1398     uint_t         gotents;

1400     raddend = rsp->rel_raddend;
1401     if (pgnp && (pgnp->gn_raddend == raddend) && (pgnp->gn_gotref == gref))
1402         return (1);

1404     if ((gref == GOT_REF_TLSD) || (gref == GOT_REF_TLSLD))
1405         gotents = 2;
1406     else
1407         gotents = 1;

1409     gn.gn_raddend = raddend;
1410     gn.gn_gotndx = ofl->ofl_gotcnt;
1411     gn.gn_gotref = gref;

1413     ofl->ofl_gotcnt += gotents;

1415     if (gref == GOT_REF_TLSLD) {
1416         if (ofl->ofl_tlsldgotndx == NULL) {
1417             if ((gnp = libld_malloc(sizeof (Gotndx))) == NULL)
1418                 return (S_ERROR);
1419             (void) memcpy(gnp, &gn, sizeof (Gotndx));
1420             ofl->ofl_tlsldgotndx = gnp;
1421         }
1422     }
1423 }

```

```

1421         }
1422         return (1);
1423     }
1425     idx = 0;
1426     for (ALIST_TRAVERSE(*alpp, idx, gnp)) {
1427         if (gnp->gn_addend > raddend)
1428             break;
1429     }
1431 /*
1432 * GOT indexes are maintained on an Alist, where there is typically
1433 * only one index. The usage of this list is to scan the list to find
1434 * an index, and then apply that index immediately to a relocation.
1435 * Thus there are no external references to these GOT index structures
1436 * that can be compromised by the Alist being reallocated.
1437 */
1438 if (alist_insert(alpp, &gn, sizeof (Gotndx),
1439     AL_CNT_SDP_GOT, idx) == NULL)
1440     return (S_ERROR);
1442
1443 }
1445 static void
1446 ld_assign_plt_ndx(Sym_desc * sdp, Ofl_desc *ofl)
1447 {
1448     sdp->sd_aux->sa_PLTndx = 1 + ofl->ofl_pltcnt++;
1449     sdp->sd_aux->sa_PLTGOTndx = ofl->ofl_gotcnt++;
1450     ofl->ofl_flags |= FLG_OF_BLDGOT;
1451 }
1453 static uchar_t plt0_template[M_PLT_ENTSIZE] = {
1454 /* 0x00 PUSHQ GOT+8(%rip) */ 0xff, 0x35, 0x00, 0x00, 0x00, 0x00,
1455 /* 0x06 JMP    *GOT+16(%rip) */ 0xff, 0x25, 0x00, 0x00, 0x00, 0x00,
1456 /* 0x0c NOP */ 0x90,
1457 /* 0x0d NOP */ 0x90,
1458 /* 0x0e NOP */ 0x90,
1459 /* 0x0f NOP */ 0x90
1460 };
1462 /*
1463 * Initializes .got[0] with the _DYNAMIC symbol value.
1464 */
1465 static uintptr_t
1466 ld_fillin_gotplt(Ofl_desc *ofl)
1467 {
1468     int      bswap = (ofl->ofl_flags1 & FLG_OF1_ENCDIFF) != 0;
1469
1470     if (ofl->ofl_osgot) {
1471         Sym_desc      *sdp;
1472
1473         if ((sdp = ld_sym_find(MSG_ORIG(MSG_SYM_DYNAMIC_U),
1474             SYM_NOHASH, NULL, ofl)) != NULL) {
1475             uchar_t *genptr;
1476
1477             genptr = ((uchar_t *)ofl->ofl_osgot->os_outdata->d_buf +
1478                         (M_GOT_XDYNAMIC * M_GOT_ENTSIZE));
1479             /* LINTED */
1480             *(Xword *)genptr = sdp->sd_sym->st_value;
1481             if (bswap)
1482                 /* LINTED */
1483                 *(Xword *)genptr =
1484                     /* LINTED */
1485                     ld_bswap_Xword(*(Xword *)genptr);
1486         }
1487     }

```

```

1487     }
1488
1489     /*
1490      * Fill in the reserved slot in the procedure linkage table the first
1491      * entry is:
1492      *   0x00 PUSHQ      GOT+8(%rip)      # GOT[1]
1493      *   0x06 JMP       *GOT+16(%rip)    # GOT[2]
1494      *   0x0c NOP
1495      *   0x0d NOP
1496      *   0x0e NOP
1497      *   0x0f NOP
1498
1499     if ((ofl->ofl_flags & FLG_OF_DYNAMIC) && ofl->ofl_osplt) {
1500         uchar_t *pltent;
1501         Xword    val1;
1502
1503         pltent = (uchar_t *)ofl->ofl_osplt->os_outdata->d_buf;
1504         bcopy(plt0_template, pltent, sizeof (plt0_template));
1505
1506         /*
1507          * If '-z noreloc' is specified - skip the do_reloc_ld
1508          * stage.
1509          */
1510         if (!OFL_DO_RELOC(ofl))
1511             return (1);
1512
1513         /*
1514          * filin:
1515          *   PUSHQ GOT + 8(%rip)
1516          *
1517          * Note: 0x06 below represents the offset to the
1518          * next instruction - which is what %rip will
1519          * be pointing at.
1520          */
1521         val1 = (ofl->ofl_osgot->os_shdr->sh_addr) +
1522             (M_GOT_XLINKMAP * M_GOT_ENTSIZE) -
1523             ofl->ofl_osplt->os_shdr->sh_addr - 0x06;
1524
1525         if (do_reloc_ld(&rdesc_r_amd64_gotpcrel, &pltent[0x02],
1526             &val1, syn_rdesc_sym_name, MSG_ORIG(MSG_SPECFIL_PLTENT),
1527             bswap, ofl->ofl_lml) == 0) {
1528             ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_PLT_PLT0FAIL));
1529             return (S_ERROR);
1530         }
1531
1532         /*
1533          * filin:
1534          *   JMP *GOT+16(%rip)
1535          */
1536         val1 = (ofl->ofl_osgot->os_shdr->sh_addr) +
1537             (M_GOT_XRTLD * M_GOT_ENTSIZE) -
1538             ofl->ofl_osplt->os_shdr->sh_addr - 0x0c;
1539
1540         if (do_reloc_ld(&rdesc_r_amd64_gotpcrel, &pltent[0x08],
1541             &val1, syn_rdesc_sym_name, MSG_ORIG(MSG_SPECFIL_PLTENT),
1542             bswap, ofl->ofl_lml) == 0) {
1543             ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_PLT_PLT0FAIL));
1544             return (S_ERROR);
1545         }
1546     }
1547
1548 }
1549

```

```

1553 /*
1554 * Template for generating "void (*)(void)" function
1555 */
1556 static const uchar_t nullfunc_tmpl[] = { /* amd64 */
1557 /* 0x00 */ 0x55, /* pushq %rbp */
1558 /* 0x01 */ 0x48, 0x8b, 0xec, /* movq %rsp,%rbp */
1559 /* 0x04 */ 0x48, 0x8b, 0xe5, /* movq %rbp,%rsp */
1560 /* 0x07 */ 0xd, /* popq %rbp */
1561 /* 0x08 */ 0xc3, /* ret */
1562 };
1563
1564 /*
1565 * Function used to provide fill padding in SHF_EXECINSTR sections
1566 *
1567 *
1568 * entry:
1569 *
1570 *      base - base address of section being filled
1571 *      offset - starting offset for fill within memory referenced by base
1572 *      cnt - # bytes to be filled
1573 *
1574 * exit:
1575 *      The fill has been completed.
1576 */
1577 static void
1578 execfill(void *base, off_t off, size_t cnt)
1579 {
1580     /*
1581     * 0x90 is an X86 NOP instruction in both 32 and 64-bit worlds.
1582     * There are no alignment constraints.
1583     */
1584     (void) memset(off + (char *)base, 0x90, cnt);
1585 }
1586
1587 /*
1588 * Return the ld_targ definition for this target.
1589 */
1590
1591 const Target *
1592 ld_targ_init_x86(void)
1593 {
1594     static const Target _ld_targ = {
1595         /* Target_mach */
1596         M_MACH, /* m_mach */
1597         M_MACHPLUS, /* m_machplus */
1598         M_FLAGSPLUS, /* m_flagsplus */
1599         M_CLASS, /* m_class */
1600         M_DATA, /* m_data */
1601
1602         M_SEGM_ALIGN, /* m_segm_align */
1603         M_SEGM_ORIGIN, /* m_segm_origin */
1604         M_SEGM_AORIGIN, /* m_segm_aorigin */
1605         M_DATASEG_PERM, /* m_dataseg_perm */
1606         M_STACK_PERM, /* m_stack_perm */
1607         M_WORD_ALIGN, /* m_word_align */
1608         MSG_ORIG(MSG_PTH_RTLD_AMD64), /* m_def_interp */
1609
1610         /* Relocation type codes */
1611         M_R_ARRAYADDR, /* m_r_arrayaddr */
1612         M_R_COPY, /* m_r_copy */
1613         M_R_GLOB_DAT, /* m_r_glob_dat */
1614         M_R JMP_SLOT, /* m_r_jmp_slot */
1615         M_R_NUM, /* m_r_num */
1616         M_R_NONE, /* m_r_none */
1617         M_R_RELATIVE, /* m_r_relative */
1618         M_R REGISTER, /* m_r_register */

```

```

1620 /* Relocation related constants */
1621 M_REL_DT_COUNT, /* m_rel_dt_count */
1622 M_REL_DT_ENT, /* m_rel_dt_ent */
1623 M_REL_DT_SIZE, /* m_rel_dt_size */
1624 M_REL_DT_TYPE, /* m_rel_dt_type */
1625 M_REL_SHT_TYPE, /* m_rel_sht_type */

1626 /* GOT related constants */
1627 M_GOT_ENTSIZE, /* m_got_entsize */
1628 M_GOT_XNumber, /* m_got_xnumber */

1629 /* PLT related constants */
1630 M_PLT_ALIGN, /* m_plt_align */
1631 M_PLT_ENTSIZE, /* m_plt_entsize */
1632 M_PLT_RESERVSZ, /* m_plt_reservsz */
1633 M_PLT_SHF_FLAGS, /* m_plt_shf_flags */

1634 /* Section type of .eh_frame/.eh_frame_hdr sections */
1635 SHT_AMD64_UNWIND, /* m_sht_unwind */

1636 /* Section type of .eh_frame/.eh_frame_hdr sections */
1637 SHT_AM32_UNWIND, /* m_sht_unwind */

1638 /* DT related constants */
1639 M_DT_REGISTER, /* m_dt_register */

1640 },
1641 },
1642 /* Target_machid */
1643 M_ID_ARRAY, /* id_array */
1644 M_ID_BSS, /* id_bss */
1645 M_ID_CAP, /* id_cap */
1646 M_ID_CAPINFO, /* id_capinfo */
1647 M_ID_CAPCHAIN, /* id_capchain */
1648 M_ID_DATA, /* id_data */
1649 M_ID_DYNAMIC, /* id_dynamic */
1650 M_ID_DYNSORT, /* id_dynsort */
1651 M_ID_DYNSTR, /* id_dynstr */
1652 M_ID_DYNSYM, /* id_dynsym */
1653 M_ID_DYNSYM_NDX, /* id_dynsym_ndx */
1654 M_ID_GOT, /* id_got */
1655 M_ID_UNKNOWN, /* id_gotdata (unused) */
1656 M_ID_HASH, /* id_hash */
1657 M_ID_INTERP, /* id_interp */
1658 M_ID_LBSS, /* id_lbss */
1659 M_ID_LDYNSYM, /* id_ldynsym */
1660 M_ID_NOTE, /* id_note */
1661 M_ID_NULL, /* id_null */
1662 M_ID_PLT, /* id_plt */
1663 M_ID_REL, /* id_rel */
1664 M_ID_STRTAB, /* id_strtab */
1665 M_ID_SYMINFO, /* id_syminfo */
1666 M_ID_SYMTAB, /* id_symtab */
1667 M_ID_SYMTAB_NDX, /* id_symtab_ndx */
1668 M_ID_TEXT, /* id_text */
1669 M_ID_TLS, /* id_tls */
1670 M_ID_TLSBSS, /* id_tlbss */
1671 M_ID_UNKNOWN, /* id_unknown */
1672 M_ID_UNWIND, /* id_unwind */
1673 M_ID_UNWINDHDR, /* id_unwindhdr */
1674 M_ID_USER, /* id_user */
1675 M_ID_VERSION, /* id_version */

1676 },
1677 /* Target_nullfunc */
1678 nullfunc_tmpl, /* nf_template */
1679 sizeof (nullfunc_tmpl), /* nf_size */

1680 },
1681 /* Target_fillfunc */
1682 execfill, /* ff_execfill */

1683 },
1684 /* Target_machrel */

```

```
1685         reloc_table,
1686
1687     ld_init_rel,           /* mr_init_rel */
1688     ld_mach_eflags,       /* mr_mach_eflags */
1689     ld_mach_make_dynamic, /* mr_mach_make_dynamic */
1690     ld_mach_update_odynamic, /* mr_mach_update_odynamic */
1691     ld_calc_plt_addr,    /* mr_calc_plt_addr */
1692     ld_perform_outreloc, /* mr_perform_outreloc */
1693     ld_do_activerelocs,  /* mr_do_activerelocs */
1694     ld_add_outrel,        /* mr_add_outrel */
1695     NULL,                /* mr_reloc_register */
1696     ld_reloc_local,      /* mr_reloc_local */
1697     NULL,                /* mr_reloc_GOTOP */
1698     ld_reloc_TLS,         /* mr_reloc_TLS */
1699     NULL,                /* mr_assign_got */
1700     ld_find_got_ndx,     /* mr_find_got_ndx */
1701     ld_calc_got_offset,   /* mr_calc_got_offset */
1702     ld_assign_got_ndx,    /* mr_assign_got_ndx */
1703     ld_assign_plt_ndx,    /* mr_assign_plt_ndx */
1704     NULL,                /* mr_allocate_got */
1705     ld_fillin_gotpplt,    /* mr_fillin_gotpplt */
1706 },
1707 {
1708     /* Target_machsym */
1709     NULL,                /* ms_reg_check */
1710     NULL,                /* ms_mach_sym_typecheck */
1711     NULL,                /* ms_is_regsym */
1712     NULL,                /* ms_reg_find */
1713     NULL,                /* ms_reg_enter */
1714 };
1715
1716 return (&_ld_targ);
1717 }
```

```

new/usr/src/cmd/sgs/test/ld/x64/tls/ie/Makefile
*****
2522 Mon Nov 5 20:40:46 2012
new/usr/src/cmd/sgs/test/ld/x64/tls/ie/Makefile
3337 x64 link-editor is painfully literal-minded about TLS
*****
1 #
2 # This file and its contents are supplied under the terms of the
3 # Common Development and Distribution License (" CDDL"), version 1.0.
4 # You may only use this file in accordance with the terms of version
5 # 1.0 of the CDDL.
6 #
7 # A full copy of the text of the CDDL should have accompanied this
8 # source. A copy of the CDDL is also available via the Internet at
9 # http://www.illumos.org/license/CDDL.
10 #

12 # Copyright 2012, Richard Lowe.

14 include $(SRC)/Makefile.master

16 # We have to use GCC, and only GCC. The best way is to ask cw(1) which GCC to u
17 CC_CMD = $(ONBLD_TOOLS)/bin/$(MACH)/cw _gcc _compiler
18 CC = $(CC_CMD):sh
19 CFLAGS = -O1 -m64

21 LINK.c = env LD_ALTEXEC=$(PROTO)/usr/bin/amd64/ld $(CC) $(CFLAGS) -o $@ $^
22 COMPILE.c = $(CC) $(CFLAGS) -c -o $@ $^
23 COMPILE.s = $(CC) $(CFLAGS) -c -o $@ $^

25 .KEEP_STATE:

27 install default: all

29 .c.o:
30     $(COMPILE.c)

32 .s.o:
33     $(COMPILE.s)

35 # A basic use of TLS that uses the movq m/r --> movq i/r variant
36 PROGS += style2
37 STYLE2OBJS = style2.o
38 style2: $(STYLE2OBJS)
39     $(LINK.c)

41 # A copy of style2 that uses %r13 in the TLS sequence, and thus excercises the
42 # REX transitions of the movq mem,reg -> movq imm,reg variant.
43 PROGS += style2-with-r13
44 STYLE2R13OBJS = style2-with-r13.o
45 style2-with-r13: $(STYLE2R13OBJS)
46     $(LINK.c)

48 # A copy of style2 that uses %r12 in the TLS sequence, so we can verify that
49 # it is _not_ special to this variant
50 PROGS += style2-with-r12
51 STYLE2R12OBJS = style2-with-r12.o
52 style2-with-r12: $(STYLE2R12OBJS)
53     $(LINK.c)

55 # A copy of style2 that has a R_AMD64_GOTTPOFF relocation with a bad insn sequen
56 STYLE2BADNESSOBJS = style2-with-badness.o
57 style2-with-badness: $(STYLE2BADNESSOBJS)
58     -$(LINK.c)

60 # A basic use of TLS that uses the addq mem/reg --> leaq mem,reg variant
61 PROGS += style1

```

```

1
new/usr/src/cmd/sgs/test/ld/x64/tls/ie/Makefile
*****
62 STYLE1OBJS = style1-main.o style1-func.o
63 style1: $(STYLE1OBJS)
64     $(LINK.c)

66 # A copy of style1-func that uses %r13 in the TLS sequence and thus excercises
67 # the REX transitions. of the addq mem,reg --> leaq mem,reg variant
68 PROGS += style1-with-r13
69 STYLE1R13OBJS = style1-main.o style1-func-with-r13.o
70 style1-with-r13: $(STYLE1R13OBJS)
71     $(LINK.c)

73 # A copy of style1-func that uses %r12 to test the addq mem,reg --> addq imm,reg
74 PROGS += style1-with-r12
75 STYLE1R12OBJS = style1-main.o style1-func-with-r12.o
76 style1-with-r12: $(STYLE1R12OBJS)
77     $(LINK.c)

79 all: $(PROGS)

81 clobber clean:
82     rm -f $(PROGS) $(STYLE1OBJS) $(STYLE1R13OBJS) $(STYLE1R12OBJS) \
83             $(STYLE2OBJS) $(STYLE2R13OBJS) $(STYLE2R12OBJS) $(STYLE2BADNESSOBJS)

85 fail: style2-with-badness FRC

87 FRC:
88 #endif /* ! codereview */


```

```
new/usr/src/cmd/sgs/test/ld/x64/tls/ie/README
```

```
1
```

```
*****
```

```
306 Mon Nov 5 20:40:47 2012
```

```
new/usr/src/cmd/sgs/test/ld/x64/tls/ie/README
```

```
3337 x64 link-editor is painfully literal-minded about TLS
```

```
*****
```

```
1 This tests the x64 link-editor's handling of Initial Executable TLS sequences.
```

```
3 The original C source files are in orig/ but unused, since we need to avoid  
4 any changes to the compiler influencing our tests.
```

```
6      % ksh test.sh /path/to/proto/root  
7      pass: addq-->leaq 1
```

```
8      ...  
9      pass: bad insn sequence
```

```
10 #endif /* ! codereview */
```

```
*****
```

```
567 Mon Nov 5 20:40:47 2012
```

```
new/usr/src/cmd/sgs/test/ld/x64/tls/ie/orig/style1-func.c
```

```
3337 x64 link-editor is painfully literal-minded about TLS
```

```
*****
```

```
1 /*  
2  * This file and its contents are supplied under the terms of the  
3  * Common Development and Distribution License ("CDDL"), version 1.0.  
4  * You may only use this file in accordance with the terms of version  
5  * 1.0 of the CDDL.  
6  *  
7  * A full copy of the text of the CDDL should have accompanied this  
8  * source. A copy of the CDDL is also available via the Internet at  
9  * http://www.illumos.org/license/CDDL.  
10 */  
  
12 /*  
13  * Copyright 2012, Richard Lowe.  
14 */  
  
16 #include <stdio.h>  
  
18 extern __thread char *foo, *bar;  
  
20 void  
21 func()  
22 {  
23     printf("foo: %p bar: %p\n", &foo, &bar);  
24 }  
25 #endif /* ! codereview */
```

```
*****
```

```
636 Mon Nov 5 20:40:47 2012
```

```
new/usr/src/cmd/sgs/test/ld/x64/tls/ie/orig/style1-main.c
```

```
3337 x64 link-editor is painfully literal-minded about TLS
```

```
*****
```

```
1 /*  
2  * This file and its contents are supplied under the terms of the  
3  * Common Development and Distribution License ("CDDL"), version 1.0.  
4  * You may only use this file in accordance with the terms of version  
5  * 1.0 of the CDDL.  
6  *  
7  * A full copy of the text of the CDDL should have accompanied this  
8  * source. A copy of the CDDL is also available via the Internet at  
9  * http://www.illumos.org/license/CDDL.  
10 */  
  
12 /*  
13  * Copyright 2012, Richard Lowe.  
14 */  
  
16 #include <stdio.h>  
  
18 extern void func();  
  
20 __thread char *foo = "foo";  
21 __thread char *bar = "bar";  
  
23 int  
24 main(void)  
25 {  
26     printf("foo: %p bar: %p\n", &foo, &bar);  
27     func();  
28     return (0);  
29 }  
30 #endif /* ! codereview */
```

```
new/usr/src/cmd/sgs/test/ld/x64/tls/ie/orig/style2.c
```

```
1
```

```
*****
```

```
607 Mon Nov 5 20:40:47 2012
```

```
new/usr/src/cmd/sgs/test/ld/x64/tls/ie/orig/style2.c
```

```
3337 x64 link-editor is painfully literal-minded about TLS
```

```
*****
```

```
1 /*  
2  * This file and its contents are supplied under the terms of the  
3  * Common Development and Distribution License ("CDDL"), version 1.0.  
4  * You may only use this file in accordance with the terms of version  
5  * 1.0 of the CDDL.  
6  *  
7  * A full copy of the text of the CDDL should have accompanied this  
8  * source. A copy of the CDDL is also available via the Internet at  
9  * http://www.illumos.org/license/CDDL.  
10 */  
  
12 /*  
13  * Copyright 2012, Richard Lowe.  
14 */  
  
16 #include <stdio.h>  
  
18 __thread char *foo __attribute__((tls_model("initial-exec"))) = "foo";  
  
20 int  
21 main(void)  
22 {  
23     printf("foo: %p\n", &foo);  
24     return (0);  
25 }  
26 #endif /* ! codereview */
```

```
new/usr/src/cmd/sgs/test/ld/x64/tls/ie/style1-func-with-r12.s
```

```
1
```

```
*****  
847 Mon Nov  5 20:40:47 2012  
new/usr/src/cmd/sgs/test/ld/x64/tls/ie/style1-func-with-r12.s  
3337 x64 link-editor is painfully literal-minded about TLS  
*****
```

```
1 /*  
2  * This file and its contents are supplied under the terms of the  
3  * Common Development and Distribution License ("CDDL"), version 1.0.  
4  * You may only use this file in accordance with the terms of version  
5  * 1.0 of the CDDL.  
6  *  
7  * A full copy of the text of the CDDL should have accompanied this  
8  * source. A copy of the CDDL is also available via the Internet at  
9  * http://www.illumos.org/license/CDDL.  
10 */  
  
12 /*  
13  * Copyright 2012, Richard Lowe.  
14 */  
  
16     .section      .rodata.str1.1,"aMS",@progbits,1  
17 .LC0:    .string "foo: %p bar: %p\n"  
18         .text  
20 .globl func  
21     .type   func, @function  
22 func:  
23 .LFB0:  
24     pushq  %rbp  
25 .LCFI0:  
26     movq  %rsp, %rbp  
27 .LCFI1:  
28     movq  %fs:0, %r12  
29     movq  %r12, %rdx  
30     addq  bar@GOTTPOFF(%rip), %rdx  
31     addq  foo@GOTTPOFF(%rip), %r12  
32     movq  %r12, %rsi  
33     movl  $.LC0, %edi  
34     movl  $0, %eax  
35     call   printf  
36     leave  
37     ret  
38 .LFE0:    .size   func, .-func  
39 #endif /* ! codereview */
```

```
new/usr/src/cmd/sgs/test/ld/x64/tls/ie/style1-func-with-r13.s
```

```
1
```

```
*****  
841 Mon Nov 5 20:40:48 2012  
new/usr/src/cmd/sgs/test/ld/x64/tls/ie/style1-func-with-r13.s  
3337 x64 link-editor is painfully literal-minded about TLS  
*****
```

```
1 /*  
2  * This file and its contents are supplied under the terms of the  
3  * Common Development and Distribution License ("CDDL"), version 1.0.  
4  * You may only use this file in accordance with the terms of version  
5  * 1.0 of the CDDL.  
6  *  
7  * A full copy of the text of the CDDL should have accompanied this  
8  * source. A copy of the CDDL is also available via the Internet at  
9  * http://www.illumos.org/license/CDDL.  
10 */  
  
12 /*  
13  * Copyright 2012, Richard Lowe.  
14 */  
15 .section      .rodata.str1.1,"aMS",@progbits,1  
16 .LC0:  
17     .string "foo: %p bar: %p\n"  
18 .text  
19 .globl func  
20     .type   func, @function  
21 func:  
22 .LFB0:  
23     pushq  %rbp  
24 .LCFI0:  
25     movq  %rsp, %rbp  
26 .LCFI1:  
27     movq  %fs:0, %r13  
28     movq  %r13, %rdx  
29     addq  bar@GOTPOFF(%rip), %rdx  
30     addq  foo@GOTPOFF(%rip), %r13  
31     movq  %r13, %rsi  
32     movl  $.LC0, %edi  
33     movl  $0, %eax  
34     call   printf  
35     leave  
36     ret  
37 .LFE0:  
38     .size   func, .-func  
39 #endif /* ! codereview */
```

```
*****
```

```
815 Mon Nov 5 20:40:48 2012
```

```
new/usr/src/cmd/sgs/test/ld/x64/tls/ie/style1-func.s
3337 x64 link-editor is painfully literal-minded about TLS
```

```
*****
```

```
1 /*
2  * This file and its contents are supplied under the terms of the
3  * CDDL License. Use, distribution, and modification of this file
4  * in source and binary forms, with or without modification, are
5  * permitted provided that the following conditions are met:
6  *
7  * 1. Redistributions of source code must retain the above copyright
8  *    notice, this list of conditions and the following disclaimer.
9  * 2. Redistributions in binary form must reproduce the above copyright
10   *    notice, this list of conditions and the following disclaimer in
11   *    the documentation and/or other materials provided with the
12   *    distribution.
13   * 3. Neither the name of the copyright holder nor the names of its
14   *    contributors may be used to endorse or promote products derived
15   *    from this software without specific prior written permission.
16   *
17   * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
18   * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
19   * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
20   * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
21   * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
22   * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
23   * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
24   * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
25   * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
26   * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
27   * THE POSSIBILITY OF SUCH DAMAGE.
28 */
29
30 .section      .rodata.str1.1,"aMS",@progbits,1
31 .LC0:         .string "foo: %p bar: %p\n"
32 .text
33 .globl func
34 .type  func, @function
35 func:
36 .LFB0:
37     pushq  %rbp
38 .LCFI0:
39     movq  %rsp, %rbp
40 .LCFI1:
41     movq  %fs:0, %rsi
42     movq  %rsi, %rdx
43     addq  bar@GOTPOFF(%rip), %rdx
44     addq  foo@GOTPOFF(%rip), %rsi
45     movl  $.LC0, %edi
46     movl  $0, %eax
47     call   printf
48     leave
49     ret
50 .LFE0:
51     .size  func, .-func
52 #endif /* ! codereview */
```

```
*****
```

```
1114 Mon Nov  5 20:40:48 2012
```

```
new/usr/src/cmd/sgs/test/ld/x64/tls/ie/style1-main.s
3337 x64 link-editor is painfully literal-minded about TLS
*****
```

```
1 /*
2  * This file and its contents are supplied under the terms of the
3  * Common Development and Distribution License ("CDDL"), version 1.0.
4  * You may only use this file in accordance with the terms of version
5  * 1.0 of the CDDL.
6 *
7 * A full copy of the text of the CDDL should have accompanied this
8 * source. A copy of the CDDL is also available via the Internet at
9 * http://www.illumos.org/license/CDDL.
10 */

12 /*
13 * Copyright 2012, Richard Lowe.
14 */

16     .section      .rodata.str1.1,"aMS",@progbits,1
17 .LC0:
18     .string "foo: %p bar: %p\n"
19     .text
20 .globl main
21     .type    main, @function
22 main:
23 .LFB0:
24     pushq   %rbp
25 .LCFI0:
26     movq   %rsp, %rbp
27 .LCFI1:
28     movq   %fs:0, %rsi
29     leaq   bar@TPOFF(%rsi), %rdx
30     addq   $foo@TPOFF, %rsi
31     movl   $.LC0, %edi
32     movl   $0, %eax
33     call   printf
34     movl   $0, %eax
35     call   func
36     movl   $0, %eax
37     leave
38     ret
39 .LFE0:
40     .size   main, .-main
41 .globl foo
42     .section      .rodata.str1.1
43 .LC1:
44     .string "foo"
45     .section      .tdata,"awT",@progbits
46     .align 8
47     .type    foo, @object
48     .size   foo, 8
49 foo:
50     .quad   .LC1
51 .globl bar
52     .section      .rodata.str1.1
53 .LC2:
54     .string "bar"
55     .section      .tdata
56     .align 8
57     .type    bar, @object
58     .size   bar, 8
59 bar:
60     .quad   .LC2
61 #endif /* ! codereview */
```

```
*****
```

```
926 Mon Nov 5 20:40:48 2012
```

```
new/usr/src/cmd/sgs/test/ld/x64/tls/ie/style2-with-badness.s
```

```
3337 x64 link-editor is painfully literal-minded about TLS
```

```
*****
```

```
1 /*  
2  * This file and its contents are supplied under the terms of the  
3  * Common Development and Distribution License ("CDDL"), version 1.0.  
4  * You may only use this file in accordance with the terms of version  
5  * 1.0 of the CDDL.  
6  *  
7  * A full copy of the text of the CDDL should have accompanied this  
8  * source. A copy of the CDDL is also available via the Internet at  
9  * http://www.illumos.org/license/CDDL.  
10 */  
  
12 /*  
13  * Copyright 2012, Richard Lowe.  
14 */  
  
16     .section      .rodata.str1.1,"aMS",@progbits,1  
17 .LC0:  
18     .string "foo: %p\n"  
19     .text  
20 .globl main  
21     .type   main, @function  
22 main:  
23 .LFB0:  
24     pushq  %rbp  
25 .LCFI0:  
26     movq  %rsp, %rbp  
27 .LCFI1:  
28     leaq    foo@GOTTPOFF(%rip), %rsi  
29     addq    %fs:0, %rsi  
30     movl    $.LC0, %edi  
31     movl    $0, %eax  
32     call    printf  
33     movl    $0, %eax  
34     leave  
35     ret  
36 .LFE0:  
37     .size   main, .-main  
38 .globl foo  
39     .section      .rodata.str1.1  
40 .LC1:  
41     .string "foo"  
42     .section      .tdata,"awT",@progbits  
43     .align 8  
44     .type   foo, @object  
45     .size   foo, 8  
46 foo:  
47     .quad   .LC1  
48 #endif /* ! codereview */
```

```
new/usr/src/cmd/sgs/test/ld/x64/tls/ie/style2-with-r12.s
```

```
1
```

```
*****  
953 Mon Nov  5 20:40:48 2012  
new/usr/src/cmd/sgs/test/ld/x64/tls/ie/style2-with-r12.s  
3337 x64 link-editor is painfully literal-minded about TLS  
*****
```

```
1 /*  
2  * This file and its contents are supplied under the terms of the  
3  * Common Development and Distribution License ("CDDL"), version 1.0.  
4  * You may only use this file in accordance with the terms of version  
5  * 1.0 of the CDDL.  
6 *  
7 * A full copy of the text of the CDDL should have accompanied this  
8 * source. A copy of the CDDL is also available via the Internet at  
9 * http://www.illumos.org/license/CDDL.  
10 */  
  
12 /*  
13  * Copyright 2012, Richard Lowe.  
14 */  
  
16     .section      .rodata.str1.1,"aMS",@progbits,1  
17 .LC0:  
18     .string "foo: %p\n"  
19     .text  
20 .globl main  
21     .type   main, @function  
22 main:  
23 .LFB0:  
24     pushq  %rbp  
25 .LCFI0:  
26     movq  %rsp, %rbp  
27 .LCFI1:  
28     movq  foo@GOTPOFF(%rip), %r12  
29     addq  %fs:0, %r12  
30     movq  %r12, %rsi  
31     movl  $.LC0, %edi  
32     movl  $0, %eax  
33     call  printf  
34     movl  $0, %eax  
35     leave  
36     ret  
37 .LFE0:  
38     .size   main, .-main  
39 .globl foo  
40     .section      .rodata.str1.1  
41 .LC1:  
42     .string "foo"  
43     .section      .tdata,"awT",@progbits  
44     .align 8  
45     .type   foo, @object  
46     .size   foo, 8  
47 foo:  
48     .quad  .LC1  
49 #endif /* ! codereview */
```

```
new/usr/src/cmd/sgs/test/ld/x64/tls/ie/style2-with-r13.s
```

```
1
```

```
*****  
953 Mon Nov  5 20:40:49 2012  
new/usr/src/cmd/sgs/test/ld/x64/tls/ie/style2-with-r13.s  
3337 x64 link-editor is painfully literal-minded about TLS  
*****
```

```
1 /*  
2  * This file and its contents are supplied under the terms of the  
3  * Common Development and Distribution License ("CDDL"), version 1.0.  
4  * You may only use this file in accordance with the terms of version  
5  * 1.0 of the CDDL.  
6  *  
7  * A full copy of the text of the CDDL should have accompanied this  
8  * source. A copy of the CDDL is also available via the Internet at  
9  * http://www.illumos.org/license/CDDL.  
10 */  
  
12 /*  
13  * Copyright 2012, Richard Lowe.  
14 */  
  
16     .section      .rodata.str1.1,"aMS",@progbits,1  
17 .LC0:  
18     .string "foo: %p\n"  
19     .text  
20 .globl main  
21     .type   main, @function  
22 main:  
23 .LFB0:  
24     pushq  %rbp  
25 .LCFI0:  
26     movq  %rsp, %rbp  
27 .LCFI1:  
28     movq  foo@GOTPOFF(%rip), %r13  
29     addq  %fs:0, %r13  
30     movq  %r13, %rsi  
31     movl  $.LC0, %edi  
32     movl  $0, %eax  
33     call  printf  
34     movl  $0, %eax  
35     leave  
36     ret  
37 .LFE0:  
38     .size   main, .-main  
39 .globl foo  
40     .section      .rodata.str1.1  
41 .LC1:  
42     .string "foo"  
43     .section      .tdata,"awT",@progbits  
44     .align 8  
45     .type   foo, @object  
46     .size   foo, 8  
47 foo:  
48     .quad  .LC1  
49 #endif /* ! codereview */
```

```
*****
```

```
926 Mon Nov  5 20:40:49 2012
```

```
new/usr/src/cmd/sgs/test/ld/x64/tls/ie/style2.s
3337 x64 link-editor is painfully literal-minded about TLS
```

```
*****
```

```
1  /*
2   * This file and its contents are supplied under the terms of the
3   * Common Development and Distribution License ("CDDL"), version 1.0.
4   * You may only use this file in accordance with the terms of version
5   * 1.0 of the CDDL.
6   *
7   * A full copy of the text of the CDDL should have accompanied this
8   * source. A copy of the CDDL is also available via the Internet at
9   * http://www.illumos.org/license/CDDL.
10  */

12 /*
13  * Copyright 2012, Richard Lowe.
14  */

16     .section      .rodata.str1.1,"aMS",@progbits,1
17 .LC0:
18     .string "foo: %p\n"
19     .text
20 .globl main
21     .type    main, @function
22 main:
23 .LFB0:
24     pushq  %rbp
25 .LCFI0:
26     movq  %rsp, %rbp
27 .LCFI1:
28     movq  foo@GOTPOFF(%rip), %rsi
29     addq  %fs:0, %rsi
30     movl  $.LC0, %edi
31     movl  $0, %eax
32     call  printf
33     movl  $0, %eax
34     leave
35     ret
36 .LFE0:
37     .size   main, .-main
38 .globl foo
39     .section      .rodata.str1.1
40 .LC1:
41     .string "foo"
42     .section      .tdata,"awT",@progbits
43     .align 8
44     .type    foo, @object
45     .size   foo, 8
46 foo:
47     .quad   .LC1
48 #endif /* ! codereview */
```

```

new/usr/src/cmd/sgs/test/ld/x64/tls/ie/x64-ie-test.sh
*****
2078 Mon Nov  5 20:40:49 2012
new/usr/src/cmd/sgs/test/ld/x64/tls/ie/x64-ie-test.sh
3337 x64 link-editor is painfully literal-minded about TLS
*****
1 #!/bin/ksh
2 #
3 # This file and its contents are supplied under the terms of the
4 # Common Development and Distribution License (" CDDL"), version 1.0.
5 # You may only use this file in accordance with the terms of version
6 # 1.0 of the CDDL.
7 #
8 # A full copy of the text of the CDDL should have accompanied this
9 # source. A copy of the CDDL is also available via the Internet at
10 # http://www.illumos.org/license/CDDL.
11 #

13 # Copyright 2012, Richard Lowe.

15 function grep_test {
16     name=$1
17     pattern=$2

19     if /usr/xpg4/bin/fgrep -q "${pattern}"; then
20         print -u2 "pass: $name"
21     else
22         print -u2 "FAIL: $name"
23     fi
24 }

26 function dis_test {
27     name=${1}
28     func=${2}
29     file=${3}
30     pattern=${4}

32     dis -F${func} ${file} | grep_test "${name}" "${pattern}"
33 }

35 make PROTO="${1}"

37 dis_test "addq-->leaq 1" func style1 \
38   'func+0x10: 48 8d 92 f8 ff ff leaq    -0x8(%rdx),%rdx'
39 dis_test "addq-->leaq 2" func style1 \
40   'func+0x17: 48 8d b6 f0 ff ff leaq    -0x10(%rsi),%rsi'

42 dis_test "addq-->leaq w/REX 1" func style1-with-r13 \
43   'func+0x10: 48 8d 92 f8 ff ff leaq    -0x8(%rdx),%rdx'
44 dis_test "addq-->leaq w/REX 2" func style1-with-r13 \
45   'func+0x17: 4d 8d ad f0 ff ff addq    $-0x10(%r13),%r13'

47 dis_test "addq-->addq for SIB 1" func style1-with-r12 \
48   'func+0x10: 48 8d 92 f8 ff ff leaq    -0x8(%rdx),%rdx'
49 dis_test "addq-->addq for SIB 2" func style1-with-r12 \
50   'func+0x17: 49 81 c4 f0 ff ff addq    $-0x10,%r12    <0xfffffffffffff0>'

52 dis_test "movq-->movq" main style2 \
53   'main+0x4: 48 c7 c6 f0 ff ff movq    $-0x10,%rsi    <0xfffffffffffff0>'

55 dis_test "movq-->movq w/REX" main style2-with-r13 \
56   'main+0x4: 49 c7 c5 f0 ff ff movq    $-0x10,%r13    <0xfffffffffffff0>'

58 dis_test "movq-->movq incase of SIB" main style2-with-r12 \
59   'main+0x4: 49 c7 c4 f0 ff ff movq    $-0x10,%r12    <0xfffffffffffff0>'

61 make PROTO="${1}" fail 2>&1 | grep_test "bad insn sequence" \

```

```

1 new/usr/src/cmd/sgs/test/ld/x64/tls/ie/x64-ie-test.sh
2
62     'ld: fatal: relocation error: R_AMD64_TPOFF32: file style2-with-badness.o: sy
63 #endif /* ! codereview */
```