
PPPPPPPPPPPP AAAAAAAAA TTTTTTTTTTTTTTT CCCCCCCCCCC HHH HHH
PPPPPPPPPPPP AAAAAAAAA TTTTTTTTTTTTTTT CCCCCCCCCCC HHH HHH
PPPPPPPPPPPP AAAAAAAAA TTTTTTTTTTTTTTT CCCCCCCCCCC HHH HHH
PPP PPP AAA AAA TTT CCC HHH HHH
PPP PPP AAA AAA TTT CCC HHH HHH
PPP PPP AAA AAA TTT CCC HHH HHH
PPP PPP AAA AAA TTT CCC HHH HHH
PPP PPP AAA AAA TTT CCC HHH HHH
PPPPPPPPPPPP AAA AAA TTT CCC HHHHHHHHHHHHHHH
PPPPPPPPPPPP AAA AAA TTT CCC HHHHHHHHHHHHHHH
PPPPPPPPPPPP AAA AAA TTT CCC HHHHHHHHHHHHHHH
PPP AAAAAAAAAAAAAAA TTT CCC HHH HHH
PPP AAAAAAAAAAAAAAA TTT CCC HHH HHH
PPP AAAAAAAAAAAAAAA TTT CCC HHH HHH
PPP AAA AAA TTT CCC HHH HHH
PPP AAAAAAAAAAAAAAA TTT CCC HHH HHH
PPP AAA AAA TTT CCCCCCCCCCC HHH HHH
PPP AAA AAA TTT CCCCCCCCCCC HHH HHH
PPP AAA AAA TTT CCCCCCCCCCC HHH HHH

```
VV      VV    AAAAAA    XX      XX    OOOOOO    PPPPPPPP      SSSSSSSS
VV      VV    AAAAAA    XX      XX    OOOOOO    PPPPPPPP      SSSSSSSS
VV      VV   AA    AA   XX      XX   OO    OO   PP      PP   SS
VV      VV   AA    AA   XX      XX   OO    OO   PP      PP   SS
VV      VV   AA    AA     XX  XX     OO    OO   PP      PP   SS
VV      VV   AA    AA     XX  XX     OO    OO   PP      PP   SS
VV      VV   AA    AA       XX       OO    OO   PPPPPPPP      SSSSSS
VV      VV   AA    AA       XX       OO    OO   PPPPPPPP      SSSSSS
VV      VV   AAAAAAAAAA     XX  XX   OO    OO   PP                  SS
VV      VV   AAAAAAAAAA     XX  XX   OO    OO   PP                  SS
  VV  VV     AA    AA   XX      XX   OO    OO   PP                  SS      ....
  VV  VV     AA    AA   XX      XX   OO    OO   PP                  SS      ....
    VV       AA    AA   XX      XX     OOOOOO     PP          SSSSSSSS      ....
    VV       AA    AA   XX      XX     OOOOOO     PP          SSSSSSSS      ....
```

```
RRRRRRRR    EEEEEEEEEE    QQQQQQ
RRRRRRRR    EEEEEEEEEE    QQQQQQ
RR      RR  EE            QQ    QQ
RR      RR  EE            QQ    QQ
RR      RR  EE            QQ    QQ
RR      RR  EE            QQ    QQ
RRRRRRRR    EEEEEEEE      QQ    QQ
RRRRRRRR    EEEEEEEE      QQ    QQ
RR  RR      EE            QQ  QQ  QQ
RR  RR      EE            QQ  QQ  QQ
RR    RR    EE            QQ    QQ
RR    RR    EE            QQ    QQ
RR      RR  EEEEEEEEEE    QQQQ  QQ
RR      RR  EEEEEEEEEE    QQQQ  QQ
```

```
!  VAXOPS.REQ - OP CODE TABLE FOR VAX INSTRUCTIONS
!
! Version:      'V04-000'
```



```
!  KEVIN PAMMETT, MARCH 2, 1977.
!
!  Modified by:
!
!       V03-002 MTR0023         Mike Rhodes             9-May-1983
!               Fix OPCODE_TBL structure definition so that it will
!               automatically track changes in the size of the opcode tables.
!
!       V03-001 MTR0009         Mike Rhodes             08-Jul-1982
!               Update count of aliased intstructions defined in PATINS.B32.
!
!       04      CNH0013         Chris Hume              27-Aug-79       13:00
!               Added double byte OPcode (and aliased OPcode) support.
!
!               KDM0011         KATHLEEN D. MORSE       27-NOV-1978     10:25
!               Change BR_LG back to a 2-bit value and OP_BR_TYPE to a
!               2-bit field.
!
!  Revision History
!
!       01      30-JUN-77       KGP     -Added another AMODE literal
!                                       for PC-Displacement mode.
!       02      26-APR-78       KDM     -Added assembler directive indicator
!                                       ASM_DIR_OP
!       03      04-MAY-78       KDM     -Added number of assembler directives
!                                       in instruction table, NUM_ASM_DIR

LITERAL         ! CONTEXT INDICATORS USED TO FILL THE 4-BIT
                !  PER OPERAND FIELD OF EACH OPINFO ENTRY PER OPCODE.
```

```
                ! These numbers are the Pseudo-Binary-Logarithm (PBL) of the
                ! number of bytes appropriate to the context of an operand.

        NUL     = 0,    ! OPCODE DOES NOT HAVE THIS OPERAND.
        BYT     = 0,    ! PC-RELATIVE MODE CONSUMES 1 BYTE.
        WRD     = 1,    ! PC-RELATIVE MODE CONSUMES 2 BYTES,
        LNG     = 2,    ! PC-RELATIVE MODE CONSUMES 4 BYTES
        FLT     = 2,    ! PC-RELATIVE MODE CONSUMES 4 BYTES
        QAD     = 3,    ! PC-RELATIVE MODE CONSUMES 8 BYTES.
        DBL     = 3,    ! PC-RELATIVE MODE CONSUMES 8 BYTES.
        GRN     = 3,    ! PC-RELATIVE MODE CONSUMES 8 BYTES.
        OCT     = 4,    ! PC-RELATIVE MODE CONSUMES 16 BYTES.
        HUG     = 4,    ! PC-RELATIVE MODE CONSUMES 16 BYTES.

        ! BRANCH TYPE INDICATORS.

        NO_BR   = 0,    ! OPCODE HAS NO BRANCH TYPE OPERANDS
        BR_BY   = 1,    ! LAST OPERAND'S BYTE OPERAND SPECIFIES BRANCH DISPL.
        BR_WD   = 2,    ! LAST OPERAND'S WORD OPERAND SPECIFIES BRANCH DISPL.
        BR_LG   = 3;    ! LAST OPERAND'S LONG OPERAND SPECIFIES BRANCH DISPL.


MACRO
        !++
        !
        ! THE FOLLOWING 'OPI' MACRO IS USED TO BUILD SUCCESSIVE ENTRIES FOR
        ! THE 'OPINFO' TABLE.  EACH MACRO CALL CONTAINS THE
        ! INFO FOR 1 VAX OPCODE, AND THE ENTRIES ARE SIMPLY
        ! BUILT IN THE ORDER THAT THE MACRO CALLS ARE MADE -
        ! THE ASSUMPTION IS THAT THEY WILL BE MADE IN ORDER OF
        ! INCREASING OPCODE VALUES.  THIS IS NECESSARY BECAUSE
        ! THE TABLE IS ACCESSED BY USING A GIVEN OPCODE AS THE
        ! TABLE INDEX.
        !
        !--


OPI( NAME, NUMOP, OPC, OP1, OP2, OP3, OP4, OP5, OP6, BR_TYP ) =
%RAD50_11 NAME,
( OP1^7 OR NUMOP ),
( OP3^4 OR OP2 ),
( OP5^4 OR OP4 ),
( BR_TYP^4 OR OP6 ) %,

! MACROS TO ACCESS THE FIELDS.

        ! OPCODE MNEMONIC
OP_NAME         = 0,0,32,0 %,           ! LONGWORD CONTAINS 6 RAD50 CHARS.

        ! FIELD TO SAY HOW MANY OPERANDS
        !  THIS OPCODE HAS.  THIS IS ALWAYS FIXED.
OP_NUMOPS       = 4,0,4,1 %,                    !  NOTE THE SIGN EXTENSION.
                                                !  IT IS NECESSARY BECAUSE WE
                                                !  USE -1 TO INDICATE A
                                                !  RESERVED OPCODE.
```

```
        ! 1 FIELD FOR EACH POSSIBLE OPERAND
        OP_CONTEXT(I)   = 4,4*I,4,0 %,          !  CONTEXT FIELD FOR EACH
                                                !   POSSIBLE OPERAND.  THE MACRO
                                                !   WORKS FOR I FROM 1 TO 6,
                                                !   INCLUSIVE.

        ! BRANCH TYPE ADDRESSING FIELD
        OP_BR_TYPE      = 7,4,4,0 %;            ! 4-BIT FIELD WHICH CONTAINS
                                                ! ONE OF THE 'BRANCH TYPE'
                                                ! INDICATORS GIVEN ABOVE.

MACRO
        !++
        ! The following 'ALI' macro is used to build successive entries for
        ! the 'ALIAS' table.  Each macro call contains the OPcode to which
        ! an alias maps.  The 'OPINFO' table can then be accessed to find the
        ! relevant information on the OPcode.
        !--
        ALI( NAME, OPC ) =
        %RAD50_11 NAME, WORD( OPC ) %,

        ! The OP_NAME macro can be used for both the 'OPINFO' and the 'ALIAS'
        ! table entries.  The following macro is used to access the OPcode to
        ! which an alias maps.

        AL_OPC          = 4,0,16,0 %;

LITERAL
        OPTSIZE = 8,            ! EACH OPINFO BLOCK IS 8 BYTES LONG.
        SIZOPINFO1 = 259,       ! Number of PAT$GB_OPINFO1 entries
        SIZOPINFO2 = 256,       ! Number of PAT$GB_OPINFO2 entries
        ALTSIZE = 6,            ! Each ALIAS block is 6 bytes long.
        SIZALIAS = 53,          ! Number of PAT$GB_ALIAS entries
        NUM_ASM_DIR = 3,        ! NUMBER OF ASSEMBLER DIRECTIVES INCLUDED.
        MAXOPRNDS = 6,          ! MAXIMUM NUMBER OF OPERANDS PER INSTRUCTION.
        BITS_PER_BYTE = 8,      ! NUMBER OF BITS IN A VAX BYTE.
        AP_REG  = 12,           ! NUMBER OF PROCESSOR REGISTER, 'AP'.
        PC_REG  = 15,           ! NUMBER OF PROCESSOR REGISTER, 'PC'.

        PC_REL_MODE = 8,        ! ADDRESSING MODE: (PC)+
        AT_PC_REL_MODE = 9,     ! ADDRESSING MODE: @(PC)+
        INDEXING_MODE = 4,      ! ADDRESSING MODE: XXX[RX]

        SHORT_LIT_AMODE = 0,    ! Short literals fit right into the mode byte.
        REGISTER_AMODE  = 5,    ! Register mode addressing.
        REG_DEF_AMODE   = 6,    ! Register deferred addressing mode.
        AUTO_DEC_AMODE  = 7,    ! Auto decrement addressing mode.
        AUTO_INC_AMODE  = 8,    ! Auto Increment addressing mode.
        DISP_BYTE_AMODE = 10,   ! All of the displacement modes start from
                                !  here.  See ENC_OPERAND() IN DBGENC.B32
        DISP_LONG_AMODE = 14,
        OP_CH_SIZE = 6;         ! SIZE, IN ASCII CHARS, OF OPCODE MNEMONIC.

MACRO
        DSPL_MODE = 0,4,4,0 %,  ! ADDRESSING MODE BITS FROM THE DOMINANT MODE
                                !  BYTE OF AN OPERAND REFERENCE.
```

```
        DOM_MOD_FIELD = 0,5,2,1 %,
                                ! BITS WHICH WE PICK UP TO DIFFERENTIATE CERTAIN
                                !  TYPES OF DOMINANT MODES.  SEE DBGMAC.B32
        SHORT_LITERAL = 0,0,6,0 %,
                                ! HOW TO EXCTRACT A 'SHORT LITERAL' FROM
                                !   THE INSTRUCTION STREAM.  SEE SRM.
        AMODE   = 0,4,4,1 %,    ! BITS OF DOMINANT MODE ADDRESSING BYTE
                                !   WHICH SPECIFY THE ACTUAL MODE.
        AREG    = 0,0,4,0 %,    ! BITS OF DOMINANT MODE ADDRESSING BYTE
                                !   WHICH SPECIFY REGISTER NUMBER, ETC.
        NOT_AN_OP = -1 %,       ! OPINFO INDICATOR FOR UNASSIGNED OPCODES.
        RESERVED = 'XXX XX' %,  ! NAME OF RESERVED OPCODES.
        ASM_DIR_OP = -2 %;      ! ASSEMBLER DIRECTIVES

MACRO
        NEXT_FIELD(INDEX)       ! USED TO GET THE ADDRESS OF THE NEXT
                                !   FIELD OF A BLOCK.
                = (INDEX),0,0,0 %;

!  MACROS AND LITERALS SPECIFICALLY FOR INSTRUCTON ENCODING.
!  ('MACHINE -IN'.)

LITERAL
        BAD_OPCODE      = 1,    ! CAN'T INTERPRET THE GIVEN ASCII OPCODE.
        BAD_OPERAND     = 2,    ! UNDECODABLE OPERAND REFERENCE.
        BAD_OPRNDS      = 3,    ! WRONG NUMBER OF OPERANDS.
        INS_RESERVED    = 4;    ! GIVEN OPCODE IS RESERVED.

LITERAL
        OP_CASEB        = %X'8F',                       ! OPCODE FOR CASEB INSTRUCTION
        OP_CASEW        = %X'AF',                       ! OPCODE FOR CASEW INSTRUCTION
        OP_CASEL        = %X'CF';                       ! OPCODE FOR CASEL INSTRUCTION

! The following structure declaration selects the proper OPcode
! table by checking to see if the OPcode is a double byte OPcode.
STRUCTURE OPCODE_TBL [OPC,O,P,S,E] =
        BEGIN
        EXTERNAL
                PAT$GB_OPINFO1: BLOCKVECTOR[ SIZOPINFO1, OPTSIZE, BYTE],
                PAT$GB_OPINFO2: BLOCKVECTOR[ SIZOPINFO2, OPTSIZE, BYTE];
        IF (OPC AND %X'FF') NEQ %X'FD'
        THEN PAT$GB_OPINFO1[OPC,0,0,%BPVAL,0]           ! One byte OPcodes
        ELSE PAT$GB_OPINFO2[(OPC^-8),0,0,%BPVAL,0]      ! Two byte OPcodes
        END<P,S,E>;
```