```
.TOC    "UPT FORMAT"

;                    TOPS20                              TOPS10
;          +-----------------------------------+    +-----------------------------------+
;        0 !   Reserved                        !    ! User pg 0,,User pg 1              !
;      377 !                                   !    ! User pg 776,,User pg 777          !
;          +-----------------------------------+    +-----------------------------------+
;      400 !   Reserved                        !    ! Exec pg 340,,Exec pg 341          !
;      417 !                                   !    ! Exec pg 377,,Exec pg 377          !
;          +-----------------------------------+    +-----------------------------------+
;      420 ! Adr of LUUO block                 !
;          +-----------------------------------+
;      421 ! User arith overflow trap ins      !
;          +-----------------------------------+
;      422 ! User stack overflow trap ins      !
;          +-----------------------------------+
;      423 ! User trap 3 trap ins              !
;          +-----------------------------------+
;      424 ! MUUO flags        ! MUUO op code  !
;          +-----------------------------------+
;      425 ! MUUO old PC                       !
;          +-----------------------------------+
;      426 ! E of MUUO                         !
;          +-----------------------------------+
;      427 ! MUUO process context word         !
;          +-----------------------------------+
;      430 ! Exec no trap Muuo new PC          !
;          +-----------------------------------+
;      431 !   Reserved                        !
;      433 !                                   !
;          +-----------------------------------+
;      434 ! User no trap MUUO new PC          !
;          +-----------------------------------+
;      435 !   Reserved                        !
;      477 !                                   !
;          +-----------------------------------+
;      500 ! Page fail word                    !
;          +-----------------------------------+    +-----------------------------------+
;      501 ! Page fail flags                   !    ! Page fail old flags,,PC           !
;          +-----------------------------------+    +-----------------------------------+
;      502 ! Page fail old PC                  !    ! Page fail new PC                  !
;          +-----------------------------------+    +-----------------------------------+
;      503 ! Page fail new PC                  !    ! Reserved                          !
;          +-----------------------------------+    +-----------------------------------+
;      504 !   Reserved                        !
;      537 !                                   !
;          +-----------------------------------+
;      540 ! User section 0                    !
;          !   ...                             !
;      577 ! User section 37                   !
;          +-----------------------------------+
;      600 !   Reserved                        !
;      777 !                                   !
;          +-----------------------------------+
```

```
.TOC    "REGISTER USAGE"

;               ext             int
;       Reg 0   W1              PC FLAGS
;       Reg 1   W2              AC-OP+1
;       Reg 2   W3              MEM-OP+1
;       Reg 3   W4              EPT
;       Reg 4   W5              UPT
;       Reg 5   W6              SPT
;       Reg 6   IR              PMA
;       Reg 7   PC              TIME
;       Reg 10  E               PI REG(loads 2914 status reg)
;       Reg 11  MB              BIT4(first part done)
;       Reg 12  AC-OP           BIT12
;       Reg 13  MEM-OP          BIT17
;       Reg 14  PXCT            CO!C1
;       Reg 15  K 77            K 7777.-1
;       Reg 16  K 777           K 407777.0
;       Reg 17  K -1.0          K -1
```

```
.TOC    "CROM BITS"
        .UCODE

;In the following assume MIN6 CRM 00 to be ieftmost bit.

ALU OP 0/=<0>   ;2903 18
ALU SP FUN/=<0:8>       ;2903 18-10
        MUL=000         ;or 040, unsigned multiply
        TCM=100         ;or 140, Two's complement multiply
        S+1+C_B=200     ;increment by one or two
        SM=240          ;sign/magnitude two's complement
        TCM COR=300     ;or 340, two's complement multipiy correction
        ;=400           ;or 440, single length normalize
        DLN=500         ;or 540, double length normalize and first divide op
        TCDIV=600       ;or 640, two's complement divide
        TCDIV COR=700   ;or 740, two's complement divide correction and remainder
ALU SHIFT/=<0:3>,.DEFAULT=14    ;2903 18-15
        ASR F_B         "ALU SHIFT/0,ALU SPECIAL/0,ALU_Y/YES"   ;ASR F_Y         Q_Q     WRITE=0
        LRS F_B         "ALU SHIFT/1,ALU SPECIAL/0,ALU_Y/YES"   ;LRS R_Y         Q_Q     WRITE=1
        F_LRS Y_B       "ALU SHIFT/1,ALU SPECIAL/0,ALU_Y/NO"    ;LRS R_Y         Q_Q     WRITE=1
        (MC F)RS_[]     "MC_SION SIOO_QION,LRS F_B,B SEL/@1"
        (MN F)RS_[]     "MN_SION SIOO_QION,LRS F_B,B SEL/@1"
        ASRC_B          "ALU SHIFT/2,ALU SPECIAL/0,ALU_Y/YES"   ;F3_Y3;Y2 FQ/2_BQ
        LSRC_B          "ALU SHIFT/3,ALU SPECIAL/0,ALU_Y/YES"   ;LRS F_Y         LRS Q_Q WRITE=3
        Y_[] QRS_Q      "ALU SHIFT/3,ALU SPECIAL/0,ALU_Y/NO,B SEL/@1"   ;LRS F_ LRS Q_Q WRITE=3
        F_B             "ALU SHIFT/4,ALU SPECIAL/0,ALU_Y/YES"   ;F_Y    Q_Q     WRITE=4
        Y_B             "ALU SHIFT/4,ALU SPECIAL/0,ALU_Y/NO"    ;F_Y    Q_Q     WRITE=4
        F_Y QRS_Q       "AL  SHIFT/5,ALU SPECIAL/0,ALU_Y/YES"   ;F_Y    LRS Q_Q=5
        F_Q             "ALU SHIFT/6,ALU SPECIAL/0,ALU_Y/NO"    ;F_Y    F_Q=6
        F_Q_Y           "ALU SHIFT/6,ALU SPECIAL/0,ALU_Y/YES"   ;F_Y    F_Q=6
        F_Q_B           "ALU SHIFT/7,ALU SPECIAL/0,ALU_Y/YES"   ;F_Y    F_Q     WRITE=7
        Y_B F_Q         "ALU SHIFT/7,ALU SPECIAL/0,ALU_Y/NO"    ;F_Y    F_Q     WRITE=7
        ALS F_[]        "ALU SHIFT/10,ALU SPECIAL/0,ALU_Y/YES,0_SIOO 0_QIOO,B SEL/@1"   ;ALS F_Y
        LLS F_B         "ALU SHIFT/11,ALU SPECIAL/0,ALU_Y/YES"
        F_LSZ Y_[]      "ALU SHIFT/11,0_SIOO 0_QIOO,ALU SPECIAL/0,ALU_Y/NO,B SEL/@1"    ;LLS F_Y
        ALSC_B          "ALU SHIFT/12,ALU SPECIAL/0,ALU_Y/YES"  ;ALS F_Y         LLS Q_Q WRITE=12
        LLSC_B          "ALU SHIFT/13,ALU SPECIAL/0,ALU_Y/YES"  ;LLS F_Y         LLS Q_Q WRITE=13
        Y_[] QLS_Q      "ALU SHIFT/13,ALU SPECIAL/0,ALU_Y/NO,B SEL/@1"  ;LLS F_Y         LLS Q_Q W
        F_Y             "ALU SHIFT/14,ALU SPECIAL/0,ALU_Y/YES"  ;F_Y    Q_Q=14
        QLS_Q           "ALU SHIFT/15,ALU SPECIAL/0"    ;F_Y    LLS Q_Q=15
        F_Y QLS_Q       "ALU SHIFT/15,ALU SPECIAL/0,ALU_Y/YES"  ;F_Y    LLS Q_Q=15
                                ;SIOO_Y Q_Q     WRITE=16
                                ;F_Y    Q_Q     WRITE=17
ALU OP/=<4:7>,.DEFAULT=10       ;2903 14-11
        S-R-1+C=1
        R-S-1+C=2
        R+S+C=3
        S+C=4
        SBAR+C=5
        R+C=6
        RBAR+C=7
        LOW=10
        RBAR.AND.S=11
        R.XNOR.S=12
        R.XOR.S=13
        R.AND.S=14
        R.NOR.S=15
        R.NAND.S=16
        R.OR.S=17
```

```
ALU IO/=<8:8>,.DEFAULT=0          ;2903 IO
        Q_S             "ALU IO/1"                      ;S operand is Q reg

R SEL TYP/=<13:14>
                ;0 reserved
        REG EXT=2
        REG INT=3
        LINE #_[]       "R SEL TYP/1,ALU OP/R+C,O_CO,F_[@1]"
R SEL/=<9:14>, DEFAULT=00
        REG-EXT=02
         W1=02
         W2=06
         W3=12
         W4=16
         W5=22
         W6=26
         IR=32
         PC=36
         E=42
         MB=46
         AC-OP=52
         MEM-OP=56
         PXCT=62
         K 77=66
         K 777=72
         K -1.0=76
        REG-INT=03
         PC FLAGS=03
         AC-OP+1=07
         MEM-OP+1=13
         EPT=17
         UPT=23
         SPT=27
         PMA=33
         TIME=37
         PI REG=43
         BIT4=47
         BIT12=53
         BIT17=57
         CO!C1=63
         K 7777.-1=67
         K 407777.0=73
         K -1=77
B SEL/=<15:20>
        REG-EXT=00
         W1=00
         W2=04
         W3=10
         W4=14
         W5=20
         W6=24
         IR=30
         PC=34
         E=40
         MB=44
         AC-OP=50
         MEM-OP=54
         PXCT=60
         K 77=64
         K 777=70
```

```
         K -1.C=74
        REG-EXT-LH=01
         W1 LH=01
         W2 LH=05
         W3 LH=11
         W4 LH=15
         W5 LH=21
         W6 LH=25
         IR LH=31
         PC LH=35
         E LH=41
         MB LH=45
         AC-OP LH=51
         MEM-OP LH=55
         PXCT LH=61
         K 77 LH=65
         K 777 LH=71
         K -1.0 LH=75
        REG-EXT-RH=2
         W1 RH=02
         W2 RH=06
         W3 RH=12
         W4 RH=16
         W5 RH=22
         W6 RH=26
         IR RH=32
         PC RH=36
         E RH=42
         MB RH=46
         AC-OP RH=52
         MEM-OP RH=56
         PXCT RH=62
         K 77 RH=66
         K 777 RH=72
         K -1.0 RH=76
        REG-INT=03
         PC FLAGS=03
         AC-OP+1=07
         MEM-OP+1=13
         EPT=17
         UPT=23
         SPT=27
         PMA=33
         TIME=37
         PI REG=43
         BIT4=47
         BIT12=53
         BIT17=57
         CO!C1=63
         K 7777.-1=67
         K 407777.0=73
         K -1=77
    S SEL[]     "B SEL/@1,ALU IO/0"
CARRY OP/=<21:22>,.DEFAULT=0    ;2904 112-111
        0_CO            "CARRY OP/0"
        1_CO            "CARRY OP/1"
        CX_CO           "CARRY OP/2"    ;This is Z for ALU SP FUN stuff

SHARED OP/=<23:28>
SET FLAG/=<23:28>       ;with shared op4,5=00
```

```
        CLEAR LOCAL     "SET FLAG/10"
        SET GLOBAL      "CLEAR LOCAL"
        SET LOCAL       "SET FLAG/14"
        CLEAR USER      "SET FLAG/20"
        SET EXEC        "CLEAR USER"
        SET USER        "SET FLAG/24"
        CLEAR PAGED     "SET FLAG/30"
        SET PAGED       "SET FLAG/34"
        CLEAR RUN       "SET FLAG/40"
        SET RUN         "SET FLAG/44"
        CLEAR TOPS20    "SET FLAG/50"
        SET TOPS20      "SET FLAG/54"
        CLEAR PXCT      "SET FLAG/60"
        SET PXCT        "SET FLAG/64"
        CLEAR TRAP      "SET FLAG/70"
        SET TRAP        "SET FLAG/74"
SHIFT OP/=<23:28>       ;with shared op4,5=01
                        ; 2904 19-16 Shift linkage mux
                        ; 2904 110 comes from 2903 18
        0_SION 0_QION           "SHIFT OP/01,ALU OP 0/0"
        1_SION 1_QION           "SHIFT OP/05,ALU OP 0/0"
        0_SION SIOO_MC MN_QION  "SHIFT OP/11,ALU OP 0/0"
        1_SION SIOO_QION        "SHIFT OP/15,ALU OP 0/0"
        MC_SION SIOO_QION       "SHIFT OP/21,ALU OP 0/0"
        MN_SION SIOO_QION       "SHIFT OP/25,ALU OP 0/0"
        0_SION SIOO_QION        "SHIFT OP/31,ALU OP 0/0"
        SIOO_SION QIOO_QION     "SHIFT OP/51,ALU OP 0/0"
        QIOO_SION SIOO_QION     "SHIFT OP/75,ALU OP 0/0"

        SION_MC 0_SIOO 0_QIOO   "SHIFT OP/01,ALU OP 0/1"
        0_SIOO 0_QIOO           "SHIFT OP/11,ALU OP 0/1"
        1_SIOO 1_QIOO           "SHIFT OP/15,ALU OP 0/1"
        SION_MC QION_SIOO 0_QIOO        "SHIFT OP/21,ALU OP 0/1"
        QION_SIOO 0_QIOO        "SHIFT OP/31,ALU OP 0/1"
        SION_SIOO QION_QIOO     "SHIFT OP/51,ALU OP 0/1"
        SION_QIOO QION_SIOO     "SHIFT OP/75,ALU OP 0/1"

INT OP/=<23:28>         ;with shared op4,5=10 2914 Instruction
        MASTER CLEAR=02
        CLEAR ALL INTS=06
        CLEAR INTS FROM Y-BUS=12
        CLEAR INTS FROM MASK REGISTER=16
        CLEAR INT LAST VECTOR READ=22
        GRANT INTERRUPT "INT OP/26"
        2914 STATUS_Y   "INT OP/32"
        2914 MASK_Y     "INT OP/36"
        377_2914 MASK   "INT OP/42"
        Y_2914 STATUS   "INT OP/46"
        ;BIT CLEAR MASK REG=52
        ;BIT SET MASK REG=56
        000_2914 MASK   "INT OP/62"
        ;DISABLE INT REQ=66
        Y_2914 MASK     "INT OP/72"
        ;ENABLE INT REQ=76
SPEC SEL/=<23:28>       ;with shared op4,5=11 Load Ram/PCI adr
        Y=03            ;Low order 12 bits of Y bus are Ramfile adr
                        ; If this is an I/O adr = PCI
                        ;       Y31 is 2651 A1
                        ;       Y32 is 2651 A0
                        ;       Y33-35 are line number
```

```
        VMA=23          ;Uses low order bits from previous spec sel/page table entry
        ;AC+N=27
        XR=33
        PAGE TABLE ENTRY=77
        IW=53           ;If bits 0,1 = 11 set ILLEGAL IW.
                        ; else clear ILLEGAL IW.
                        ; If section 0 or if bits 0,1=10 set NOT GLOBAL,
                        ; else CLEAR LOCAL.
                        ; If GLOBAL load I and index reg from 1-5,
                        ; else load from 13-17.
        SEL AC+[]       "SPEC SEL/27,J.AC/@1"
        Y_RAMFILE ADR   "SPEC SEL/Y"
        Y_TTY ADR       "Y_RAMFILE ADR"

AMD2904 15-14/=<29:30>  ;2904 15-14
        MX_Y            "AMD2904 15-14/2,STATUS_Y,ALU_Y/NO"
        MX_[]           "MX_Y,Y_[@1]"
        F_Q MX_[]       "AMD2904 15-14/2,STATUS_Y,F_Q Y_[@1]"
        UX_[]           "STATUS OP/01,STATUS_Y,Y_[@1]"

STATUS OP/=<29:34>,.DEFAULT=20  ;2904 15-10
        Y_MX            "STATUS OP/00,M STATUS ENAB/YES"
        MX_UX Y_MX      "STATUS OP/00,M STATUS ENAB/YES,U STATUS ENAB/YES"
        ;MX_UX=0        ;Copy Mx to ux
        1_UX            "STATUS OP/01,U STATUS ENAB/YES"
        1_MX            "STATUS OP/01,M STATUS ENAB/YES"
        MX_UX UX_MX     "STATUS OP/02,M STATUS ENAB/YES,U STATUS ENAB/YES"
        MX_UX           "STATUS OP/02,U STATUS ENAB/YES"
        UX_MX           "STATUS OP/02,M STATUS ENAB/YES"
        0_UX            "STATUS OP/03,U STATUS ENAB/YES"
        0_MX            "STATUS OP/03,M STATUS ENAB/YES"
        IX!UOVR_UX      "STATUS OP/06,U STATUS ENAB/YES"
        IX!MOVR_MX      "STATUS OP/06,M STATUS ENAB/YES"
        0_UZ            "STATUS OP/10,U STATUS ENAB/YES"
        1_UZ            "STATUS OP/11,U STATUS ENAB/YES"
        0_UC            "STATUS OP/12,U STATUS ENAB/YES"
        1_UC            "STATUS OP/13,U STATUS ENAB/YES"
        0_UC IX_MX      "STATUS OP/12,U STATUS ENAB/YES,M STATUS ENAB/YES"
        1_UC IX_MX      "STATUS OP/13,U STATUS ENAB/YES,M STATUS ENAB/YES"
        0_UN            "STATUS OP/14,U STATUS ENAB/YES"
        1_UN            "STATUS OP/15,U STATUS ENAB/YES"
        0_UOVR          "STATUS OP/16,U STATUS ENAB/YES"
        1_UOVR          "STATUS OP/17,U STATUS ENAB/YES"
        IX_UX           "STATUS OP/20,U STATUS ENAB/YES"
        IX_MX           "STATUS OP/20,M STATUS ENAB/YES"
        IX !CBAR_UX     "STATUS OP/30,U STATUS ENAB/YES"
        IX !CBAR_MX     "STATUS OP/30,M STATUS ENAB/YES"
    ;Carry operation
        MC_C            "CARRY OP/3,STATUS OP/47"
        MC_C IX_MX      "CARRY OP/3,STATUS OP/47,M STATUS ENAB/YES"
        MC_C IX_UX      "CARRY OP/3,STATUS OP/47,U STATUS ENAB/YES"
        UC_C            "CARRY OP/3,STATUS OP/0"
    ;Test conditions
        ; U is micro status; M is machine status; I is current status
        ; Z is zero; N is negative; OVR is overflow; C is carry
        ; ! is inclusive or; X is xor; & is and; E is xnor = equivalence
        (UNxUOVR)!UZ    "STATUS OP/00"
        (MNxMOVR)!MZ    "STATUS OP/40"
        MNxMOVR         "STATUS OP/42"
        (INxIOVR)!IZ    "STATUS OP/60"
```

```
        UZ              "STATUS OP/04"  ;u-Zero
        UZ IX_MX        "STATUS OP/04,M STATUS ENAB/YES"
        MZ              "STATUS OP/44"
        MZ IX_MX        "STATUS OP/44,M STATUS ENAB/YES"
        MZ IX_UX        "STATUS OP/44,U STATUS ENAB/YES"
        IZ              "STATUS OP/64"
        IZ IX_UX        "STATUS OP/64,U STATUS ENAB/YES"
        IZ IX_MX        "STATUS OP/64,M STATUS ENAB/YES"
        NOT UZ          "STATUS OP/05"
        NOT MZ          "STATUS OP/45"
        NOT IZ          "STATUS OP/65"
        UOVR            "STATUS OP/06"  ;u-overflow
        UOVR IX_MX      "STATUS OP/06,M STATUS ENAB/YES"
        MOVR            "STATUS OP/46"
        IOVR            "STATUS OP/66"
        IOVR IX_MX      "STATUS OP/66,M STATUS ENAB/YES"
        NOT UOVR        "STATUS OP/07"
        NOT MOVR        "STATUS OP/47"
        NOT IOVR        "STATUS OP/67"
        NOT IOVR IX_MX  "STATUS OP/67,M STATUS ENAB/YES"
        UC              "STATUS OP/12"  ;u-carry
        MC              "STATUS OP/52"
        IC              "STATUS OP/72"
        NOT UC          "STATUS OP/13"
        NOT MC          "STATUS OP/53"
        NOT IC          "STATUS OP/73"
        UN              "STATUS OP/36"  ;u-negative
        UN IX_MX        "STATUS OP/36,M STATUS ENAB/YES"
        UN IX_UX        "STATUS OP/36,U STATUS ENAB/YES"
        MN              "STATUS OP/56"
        MN IX_MX        "STATUS OP/56,M STATUS ENAB/YES"
        MN IX_UX        "STATUS OP/56,U STATUS ENAB/YES"
        IN              "STATUS OP/76"
        IN IX_MX        "STATUS OP/76,M STATUS ENAB/YES"
        NOT UN          "STATUS OP/37"
        NOT MN          "STATUS OP/57"
        NOT IN          "STATUS OP/77"
        UC!UZ           "STATUS OP/10"
        UC&UZ           "STATUS OP/11"
        NOT UC!UZ       "STATUS OP/34"
        UC&NOT UZ       "STATUS OP/35"
        INxorMN         "STATUS OP/16"
U STATUS ENAB/=<35:35>,.DEFAULT=1
        YES=0
        NO=1
. STATUS ENAB/=<36:36>,.DEFAULT=1
        YES=0
        NO=1

ALU_Y/=<37:37>,.DEFAULT=0       ;2903
        NO=0
        YES=1
U PROG OP/=<38:41>,.DEFAULT=16  ;2910 13-10
        JZ=0
        CJS=1
        JMAP=2
        CJP=3
        PUSH=4
        JSRP=5
        CJV=6
```

```
JRP=7
RFCT=10
RPCT=11
CRTN=12
CJPP=13 ;Conditional jump pipeline and pop
LDCT=14
TEGL=15
CONT=16
TWB=17
```

```
.TOC    "       TEST SEL"

TEST SEL/=<42:47>
        NEVER=00
        ALWAYS=01
        CT=02           ;2904 CT = MINI STATUS TEST
        NOT CT=03
        NOT INDEXED=04
        INDEXED=05
        AC.EQ.0=06
        AC.NE.0=07
        INDIRECT=10
        NOT INDIRECT=11
        NOT (I OR XR)=12
        I OR XR=13
        ILLEGAL IW=14   ;Signal from SPEC SEL/IW
                        ; = <not section 0> and <bit0,bit1=11)
        NOT ILLEGAL IW=15
        GLOBAL IW=16    ;Signal from SPEC SEL/IW
        LOCAL IW=17     ; = <not section 0> and <bit0=1>
        NO PI REQ=20    ;i.e. something wants to interrupt
        PI REQ=21
        NOT AC REF=22
        AC REF=23
        LEGAL SECTION=24
        ILLEGAL SECTION=25
        NOT SECTION 0=26
        SECTION 0=27
        CONTEXT MATCH=30        ;i.e. no match, AC ref, or not paged
        NOT CONTEXT MATCH=31
        NOT MEM FAULT=32
        MEM FAULT=33
        LOCAL=34
        GLOBAL=35
        USER=36
        EXEC=37         ;Not exec
        PAGED=40        ;PAGED and NOT AC REF
        NOT PAGED=41    ;AC REF or NOT PAGED
        RUN=42
        NOT RUN=43
        TOPS20=44
        TOPS10=45
        PXCT=46
        NOT PXCT=47
        B BIT 3=50      ;i.e. bit 3 on B port register is set
        NOT B BIT 3=51
        ;B BIT 4=52     ;i.e. bit 4 on B port register is set
        ;NOT B BIT 4=53
        B BIT 9=54
        NOT B BIT 9=55
        B BIT 18=56     ;i.e. bit 18 on B port register is set
        NOT B BIT 18=57

        ANY BUS ERROR=60
        NO BUS ERROR=61
        NOT AC LOW=62
        AC LOW=63
        ;LOCK=64
        ;NOT LOCK=65
```

TRAP=66
NOT TRAP=67
MEM EXISTS=70
NOT MEM EXISTS=71
UNPAGED OR AC=72
NOT UNPAGED OR AC=73
NOT TIMER FLAG=74
TIMER FLAG=75
;=76
;=77
NOT MF & NOT PXCT=62

```
.TOC    "       J"
J.AC/=<52:55>           ;4 bits in J field for AC+[]
J/=<48:59>,.ADDRESS,.DEFAULT=0
        U-CODE VERSION=4002     ;Version of u-code
    ;EPT offsets
        PFW=500         ;Page fail word
        ESECT=540       ;Section pointers
    ;UPT offsets
    ;Console error codes
        ERR.URC=1       ;Unrecognized command
        ERR.NWR=2       ;Not while machine is running
        ERR.NEA=3       ;Not enough arguments
        ERR.NXA=4       ;Non existent address
        ERR.MER=5       ;Memory error
    ;PCI hdw bits
        A0=010          ;PCI adr for read status or write syn/syn/dle
        A1=020          ;PCI adr for MR1/MR2
        A1!A0=030       ;PCI adr for CR
        TXEN=001        ;Transmitter enable in CR
        RXEN=004        ;Receiver enable in CR
    ;Ramfile offsets
        BYTE MASK=000           ;Following are "byte masks"
         PAGE # MASK=011        ;bits 23-35 (9 bits)
         10B MASK=012           ;10 bits
         12B MASK=014
         SECTION # MASK=014     ;bits 24-35 (12 bits)
         13B MASK=015
         22B MASK=026
         23B MASK=027
         24B MASK=030
         27B MASK=033
         30B MASK=036
         32B MASK=040
         JFCL MASK=040  ;bits 4-35
         33B MASK=041
;Locations 45-77 contain -1

;Set of sliding bits
        BIT0=100
         OVERFLOW=100
        BIT1=101
         CARRY 0=101
        BIT2=102
         CARRY 1=102
        BIT3=103
         FLOATING OVERFLOW=103
        BIT4=104
         FIRST PART DONE=104
        BIT5=105
         USER=105
        BIT6=106
         USER IO=106
         PCU=106        ;Previous context user
        BIT7=107
         PUBLIC=107
        BIT8=110
         ADDRESS FAILURE INHIBIT=110
        BIT9=111
         TRAP 2=111
```

```
BIT10=112
 TRAP 1=112
BIT11=113
 FLOATING UNDERFLOW=113
BIT12=114
 NO DIVIDE=114
BIT13=115
BIT14=116
BIT15=117
BIT16=120
BIT17=121
BIT18=122
BIT19=123
BIT20=124
BIT21=125
BIT22=126
BIT23=127
BIT24=130
BIT25=131
BIT26=132
BIT27=133
BIT28=134
BIT29=135
BIT30=136
BIT31=137
BIT32=140
BIT33=141
BIT34=142
BIT35=143
BIT36=144       ;Phoney location

PUSHJ FLAGS=145         ;Flags to clear on a PUSHJ =021600,,0
;UNUSED=146
;MASK12-22=147

ME RCOVR=150            ;Memory error recovery routine adr
HALT CODE=151           ;Halt code
                        ; -1 = do a console command
        HALT INS=1      ;ISP JRST 4,
        HALT CONSOLE=2  ;Console command
        HALT SI=3       ;Single instruction
        HALT PFMER=4    ;Memory errors, while handling page fault
        HALT HSBMER=5   ;Memory error while writing HSB
        HALT ADRBRK=10  ;Address break
        HALT IOPF=100   ;IO page failure.
        HALT III=101    ;Illegal interrupt instruction
        HALT DSP=1000   ;Illegal u-code dispatch
                        ; Or GLOBAL at IFETCH
        HALT HME=1001   ;Hard memory error
        HALT SUP=1005   ;Startup check
HME STS=152             ;Hard memory error status
                        ; lh 2=Bus fault line,
                        ;  3=Bus parity error into CPU
                        ; rh is status from memory
                        ;Hard memory error address
                        ;Hard memory error data
NXM STS=155             ;NXM memory error status
                        ; lh=...,rh is status from memory
                        ;Soft memory error address (ECC correctable)
                        ;Soft memory error data
```

```
        SME STS=160             ;Soft memory error status
                                ; lh=2,,rh is status from memory
                                ;Soft memory error address (ECC correctable)
                                ;Soft memory error data
        INT RCOVR=163           ;Where to go if interrupted while not running
        PF RCOVR=164            ;Routine to go to if encounter page fail

        TTYRCV BITS=177         ;Bit for each receiver which has gone off
;Here begins line block for line 0
; See PCI section for pictures
        LNOSW=200               ;Line 0 status word
                LS.PI=7         ;Mask for PI channel
                LS.CLS=10       ;Close receive buffer
                LS.XOFF=20      ;Xoff flag
                LS.XOFF.ENAB=40 ;Xoff enable
                LS.RGO=100      ;Receiver go
                LS.RDN=200      ;Receiver done
                LS.REN=400      ;Receiver enable
                ;LS.XGO=1000    ;Transmitter go
                LS.XDN=2000     ;Transmitter done
                LS.XEN=4000     ;Transmitter enable
                ;LS.RESET=10000 ;Reset line
                ;LS.DSCHG=20000 ;Dataset change
                ;LS.DSENB=40000 ;Dataset enable
                ;LS.RMER=100000 ;Receiver nxm or memory err
                ;LS.XMER=200000 ;Transmitter nxm or memory err
        LNOMDW=201              ;Mode word for CTY
        LNXMDW=1                ;Offset for mode word
        LNXSYN=2                ;Offset for synch word
        LNXCRC=3                ;Offset for CRC word
                                ; 2-17 = xmt crc calculation
                                ; 20-35 = rcv crc calculation
        LNXXHD=4                ;Offset for adr of transmit header
        LNXXCHR=5               ;Offset for (DDT mode) transmit char
                                ; non DDT mode
                                ; 0-11 count of bytes left
                                ; 12-13 byte number in word
                                ; 14-35 current word physical adr
        LNXRHD=6                ;Offset for adr of receive header
        LNXRCHR=7               ;Offset for (DDT mode) receive char
;Here begins line block for line 1
        LN1SW=210               ;Line 1 status word
;Here begins line block for line 2
        LN2SW=220               ;Line 2 status word
;Here begins line block for line 3
        LN3SW=230               ;Line 3 status word
;Here begins line block for line 4
        LN4SW=240               ;Line 4 status word
;Here begins line block for line 5
        LN5SW=250               ;Line 5 status word
;Here begins line block for line 6
        LN6SW=260               ;Line 6 status word
;Here begins line block for line 7
        LN7SW=270               ;Line 7 status word
        LNZSW=300               ;End of line block

        CMDFLG=300              ;Flags for CTY/KLINIK line
            F.O=1               ; ^O has been typed
            F.S=2               ; Xoff has been typed
            F.OIF.S=3
```

```
        ;    F.USR=BIT4          ; CTY line is in User mode
        CMDRPT=301               ;Repeat counter
        CMRPTR=302               ;Putter for command response buffer
                                 ; Byte pointer format
                                 ;       bits 0-17       byte number
                                 ;       bits 18-35      ramfile adr
                                 ; ramfile word adr
                                 ;       bit 0           unused
                                 ;       bits 1-7        1st byte = 4
                                 ;       bits 8-14       2nd byte = 3
                                 ;       bits 15-21      3rd byte = 2
                                 ;       bits 22-28      4th byte = 1
                                 ;       bits 29-35      5th byte = 0
        CMRTKR=303               ;Taker for command response buffer
                                 ; Format same as CMRPTR
        CMRBUF=304               ;Command response buffer
        CMRBUF-1=303
        CMRBUF END=337           ;End of command response buffer
        CMDPTR=340               ;Pointer to put command characters into buffer
                                 ; Format same as CMRPTR
        CMDTKR=341               ;Pointer to take command characters from buffer
                                 ; Format same as CMRPTR
                                 ; Only nonzero if a break has been typed
        CMDBUF=342               ;Command buffer
        CMDBUF-1=341
        CMDBUF END=407

        MB REFIL=441             ;Save MB here for a page refil

.IF/FTADRB
        AB IF=442                ;Address break instruction fetch
        AB WR=443                ;Address break write
        AB RD=444                ;Address break read
.ENDIF/FTADRB

        EXM ADR=445              ;Last memory adr examined or deposited
        EIS W1=446               ;Extend instruction saves W1 here
        EIS W2=447               ;Extend instruction saves W2 here
        EIS W3=450               ;Extend instruction saves W3 here
        EIS W4=451               ;Extend instruction saves W4 here
        EIS W5=452               ;Extend instruction saves W5 here
        EIS W6=453               ;Extend instruction saves W6 here
        NBBYTE1=454              ;Bits 0-8
        NBBYTE3=455              ;Bits 18-26

        X1=454                   ;Temporary location
        X2=455

.IF/DEBUG3
        POOP FLAG=520
.ENDIF/DEBUG3
        CLK W2=535
        CLK W3=536
        ME W1=537                ;Save W1 here on memory error
        ME W2=540                ;Save W2 here on memory error
        WR10 TMP=541             ;Temporary location for WR10
        HSB=553                  ;Address of halt status block
        HSB MB=554               ;Save MB here when writing HSB
        HSB PMA=555              ;Save PMA here when writing HSB
        APR FLAGS=556            ;Bits 6-13 are flags which are enabled
```

```
                                ; bits 24-31 are flags which are set
                                ; bits 33-35 is 'PR PI level
        PI IN PROG=557          ;Interrupts in progress in bits 29-35
        PI SFT REQ=560          ;Software PI requests
                                ; Bits 28-35 are software requests
                                ; bits 10-17 are tty/apr requests
        PI LVL REG=561          ;bit 0 is PI system on/off flag
                                ; bits 29-35 are levels on
        PROC REG=562            ;Processor register
                                ; For picture see SET PROC REG
        RF PROC REG=563         ;Last word we wrote in hdw PROC REG
        TIME INTERVAL=564       ;Time interval
        INTERVAL COUNTER=565    ;Count it here
        TIME BASE=566           ;Two words of time (kept in u-sec)
        TIME BASE+1=567         ;Low order word of time
        CST=571
        CSTMSK=572
        CSTDATA=573
        AC BLOCK=574            ;Current AC block
        P AC BLOCK=575          ;Previous context AC block
        PCS=576                 ;Previous context section
        AC 0 0=600
        AC 1 0=620
        AC 2 0=640
        AC 3 0=660
        AC 4 0=700
        AC 5 0=720
        AC 6 0=740
        AC 7 0=760
        PAGE TABLE-1=777
        PAGE TABLE=1000

MAP SEL/=<48:51>
        DISPATCH 1=10
         OPERAND FETCH=10
        DISPATCH 2=11
         INST EXCT=11
        DISPATCH 3=12
         OPERAND STORE=12
        DISPATCH 4=13
        ;DISPATCH 5=14
        ;DISPATCH 6=15
        ;DISPATCH 7=16
         IO 0=16
        ;DISPATCH 8=17
         MISC=17
```

```
MEM OP/=<60:62>,.DEFAULT=7
        MEM HOLD          "MEM OP/0"                          ;Don't release memory
        MEM START READ    "MEM OP/1,XFER MEM,1_SION 1_QION,ALU_Y/YES"     ;Start memory with adr fr
        MEM START WRITE   "MEM OP/2,XFER MEM,1_SION 1_QION,ALU_Y/YES"     ;Start memory with adr fr
        START IO READ     "MEM OP/3,XFER MEM,1_SION 1_QION,ALU_Y/YES"     ;Start IO read with addre
        START IO WRITE    "MEM OP/4,XFER MEM,1_SION 1_QION,ALU_Y/YES"     ;Start IO write with addr
        IO TRANSFER       "MEM OP/5,XFER MEM"                       ;Transfer IO (on MEM bus)
         IO_[]            "Y_[@1],ALU_Y/NO,IO TRANSFER,XFER MEM,IX_MX"
         ALU_IO           "ALU_Y/YES,IO TRANSFER,XFER MEM,1_SION 1_QION,IX_MX"
        MEM TRANSFER      "MEM OP/5,XFER MEM"                 ;Transfer MEM
         ALU_MEM          "ALU_Y/YES,MEM TRANSFER,XFER MEM,1_SION 1_QION,IX_MX"
         []_MEM           "[@1]_[@1],ALU_MEM"
         MEM_[]           "Y_[@1],ALU_Y/NO,MEM TRANSFER,XFER MEM,IX_MX"
         MEM_IR_[]        "Y_[@1],ALU_Y/NO,MEM OP/5,LOAD IR,IX_MX"
        ;                 "MEM OP/6"                          ;Reserved
        ALLOW DCH         "MEM OP/7"                          ;Allow data channels
Y SEL/=<63:66>,.DEFAULT=0
        XFER PIPELINE     "Y SEL/1"
            J[]_Y24-Y35   "J/@1,XFER PIPELINE,ALU_Y/NO,D SEL/J"
            FJ[]_[]       "J[@1]_Y24-Y35,Y_[@2]"                    ;bits 0-23 from F, bits 24-35 fro
            F_Q FJ[]_[]   "J[@1]_Y24-Y35,F_Q Y_[@2]"                ;bits 0-23 from F, bits 24-35 fro
            J[]_[]        "ZERO_F,FJ[@1]_[@2]"
            JS[]_[]       "ZERO_F,J[@1]_Y24-Y35,B SEL/@2,1_SION 1_QION,F_LRS Y_B" ;Bit0 or J_[]
            JN[]_[]       "ONES_F,FJ[@1]_[@2]"                      ;bits 0-23 or J[]_[]
            []J[]_[]      "[@1]_F,J[@2]_Y24-Y35,Y_[@3]"             ;bits 0-23 from R SEL, bits 24-35
            ALU_CTR       "ALU_Y/YES,XFER PIPELINE,D SEL/Y"         ;Don't need U PROG OP
            J[]_CTR       "J/@1,D SEL/J,U PROG OP/LDCT"             ;Load 2910 counter from J field
            DISP []_Y     "XFER PIPELINE,ALU_Y/NO,MAP SEL/@1,D SEL/MAP,U PROG OP/CJV,TEST SEL[illegible]VEC
        XFER MEM          "Y SEL/2"       ;Enable MEM bus transceivers
        ALU_PCI           "ALU_Y/YES,Y SEL/3"
        PCI_ALU           "ALU_Y/NO,Y SEL/3"
        ALU_RAMFILE       "ALU_Y/YES,Y SEL/4"
         []_RAMFILE       "[K -1]&[@1]_Y,ALU_RAMFILE"
        RAMFILE_Y         "ALU_Y/NO,Y SEL/4"
         RAMFILE_[]       "RAMFILE_Y,Y_[@1]"
        ALU_MR2           "Y SEL/5,ALU_Y/YES"
        STATUS_Y          "Y SEL/6"
        SWAPPER_Y         "R SEL TYP/REG EXT,ALU_Y/NO,Y SEL/7"    ;A bus swapped goes to Y
         SWAP []_[]       "SWAPPER_Y,R SEL/@1,Y_[@2]"
         SWAP []_[] F_Q   "SWAPPER_Y,R SEL/@1,Y_[@2] F_Q"
        ALU_PROC REG      "Y SEL/10,ALU_Y/YES"
        LOAD IP           "Y SEL/11"      ;This implies transfer mem

T/=<67:69>,.DEFAULT=0   ;Time
MARK1/=<70>,.DEFAULT=0
MARK2/=<71>,.DEFAULT=0

;Following are dummy fields

D SEL/=<74:75>  ;Dummy field to detect D Bus select errors
                ; 0=J_D; 1=Map or V_D; 2=Y_D
        J=1     ;J_D
        MAP=2   ;MAP_D
        Y=3     ;Y_D

ALU SPECIAL/=<76:76>    ;NONZERO IF SPECIAL ALU OP
REP/=<77:77>,.DEFAULT=0 ;Set to one if repatative instruction
NRFRD/=<78:78>,.DEFAULT=0       ;New adr for ramfile read
```

```
LCTXTM/=<79:79>,.DEFAULT=0      ;Latch context match
.IF/FTCRAM
CRAM/=<80:80>,.DEFAULT=1        ;Running in cram
.ENDIF/FTCRAM
FREE/=<81:83>,.DEFAULT=0
```

```
.TOC    "MACROS"

NOOP            "U PROG OP/CONT"

J[]_D           "J/@1,D SEL/J"
GOTO Y          "XFER PIPELINE,ALU_Y/YES,U PROG OP/CJP,TEST SEL/ALWAYS,D SEL/Y"
CALL Y          "XFER PIPELINE,ALU_Y/YES,U PROG OP/CJS,TEST SEL/ALWAYS,D SEL/Y"
GOTO []         "TEST SEL/ALWAYS,U PROG OP/CJP,J[@1]_D"
GOTOP []        "TEST SEL/ALWAYS,U PROG OP/CJPP,J[@1]_D"        ;Goto and pop stack
POP             "TEST SEL/ALWAYS,U PROG OP/TEOL"

CALL []         "TEST SEL/ALWAYS,U PROG OP/CJS,J[@1]_D"
RETURN          "TEST SEL/ALWAYS,U PROG OP/CRTN"

RFCT            "U PROG OP/RFCT"
LOOP []         "J[@1]_D,U PROG OP/RPCT"
TWB [] []       "@1,J[@2]_D,U PROG OP/TWB"
PUSH            "TEST SEL/NEVER,U PROG OP/PUSH"
PUSH J[]_CTR    "TEST SEL/ALWAYS,U PROG OP/PUSH,J[@1]_D"

IF [] THEN []           "TEST SEL/@1,U PROG OP/CJP,J[@2]_D"
IF [] THENP []          "TEST SEL/@1,U PROG OP/CJPP,J[@2]_D"    ;If branch also pop
IF [] THEN Y            "TEST SEL/@1,U PROG OP/CJP,XFER PIPELINE,ALU_Y/YES,D SEL/Y"
IF [] CALL []           "TEST SEL/@1,U PROG OP/CJS,J[@2]_D"
IF [] JSRP []           "TEST SEL/@1,U PROG OP/JSRP,J[@2]_D"
IF [] D []              "TEST SEL/@1,DISP [@2]"
IF [] RETURN            "TEST SEL/@1,U PROG OP/CRTN"
IF CT [] THEN []        "TEST SEL/CT,@1,U PROG OP/CJP,J[@2]_D"  ;CT is 2904 STATUS TEST
IF CT [] THENP []       "TEST SEL/CT,@1,U PROG OP/CJPP,J[@2]_D"
IF CT [] THEN Y         "TEST SEL/CT,@1,U PROG OP/CJP,XFER PIPELINE,ALU_Y/YES,D SEL/Y"
IF CT [] CALL []        "TEST SEL/CT,@1,U PROG OP/CJS,J[@2]_D"
IF CT [] JSRP []        "TEST SEL/CT,@1,U PROG OP/JSRP,J[@2]_D"
IF CT [] RETURN         "TEST SEL/CT,@1,U PROG OP/CRTN"
IF CT [] D []           "TEST SEL/CT,@1,DISP [@2]"
IF NOT CT [] THEN []    "TEST SEL/NOT CT,@1,U PROG OP/CJP,J[@2]_D"
IF NOT CT [] THENP []   "TEST SEL/NOT CT,@1,U PROG OP/CJPP,J[@2]_D"
IF NOT CT [] THEN Y     "TEST SEL/NOT CT,@1,U PROG OP/CJP,XFER PIPELINE,ALU_Y/YES,D SEL/Y"
IF NOT CT [] CALL []    "TEST SEL/NOT CT,@1,U PROG OP/CJS,J[@2]_D"
IF NOT CT [] JSRP []    "TEST SEL/NOT CT,@1,U PROG OP/JSRP,J[@2]_D"
IF NOT CT [] RETURN     "TEST SEL/NOT CT,@1,U PROG OP/CRTN"
IF NOT CT [] D []       "TEST SEL/NOT CT,@1,DISP [@2]"

DISP []                 "MAP SEL/@1,D SEL/MAP,U PROG OP/CJV"
DISPATCH []             "TEST SEL/ALWAYS,DISP [@1]"

J[]_RAMFILE ADR         "J[@1]_Y24-Y35,ALU_Y/NO,Y_RAMFILE ADR"

Y_[]                    "B SEL/@1,Y_B"
Y_[] F_Q                "B SEL/@1,Y_B F_Q"
F_Q Y_[]                "B SEL/@1,Y_B F_Q"
F_[]                    "B SEL/@1,F_B"
F_Q RAMFILE_[]          "F_Q Y_[@2],RAMFILE_Y"
F_Q_[]                  "B SEL/@1,F_Q_B"
ROR F_[]                "LRS F_B,SIOO_SION QIOO_QION,B SEL/@1"  ;ROR F_ Q_Q     WRITE=1
F_ROR Y_[]              "F_LRS Y_B,SIOO_SION QIOO_QION,B SEL/@1"        ;ROR F_ Q_Q     WRITE=1
F_Y LSRQ_Q              "F_Y QRS_Q,O_SION O_QION"       ;Right shift Q, shift in 0
F_Y RORQ_Q              "F_Y QRS_Q,SIOO_SION QIOO_QION" ;Right shift Q, shift in 0
LSZ F_B                 "LLS F_B,O_SIOO O_QIOO"
LSZ F_[]                "LSZ F_B,B SEL/@1"
```

```
ROL F_B                 "LLS F_B,SION_SIOO QION_QIOO"   ;LLS F_Y       O_Q     WRITE=11
ROL F_[]                "ROL F_B,B SEL/@1"              ;LLS F_Y       Q_Q     WRITE=11

ZERO_F                  "ALU OP/LOW"
ZERO_Y                  "ZERO_F,F_Y"
ZERO_[]                 "ZERO_F,F_[@1]"

ONES_F                  "ALU OP/0,ALU IO/1"
ONES_[]                 "ONES_F,F_[@1]"

BITO_[]                 "ZERO_F,LRS F_B,1_SION 1_QION,B SEL/@1"

Q+C_F                   "Q_S,ALU OP/S+C"
Q+1_F                   "1_CO,Q+C_F"
Q_F                     "Q%[K -1]_F"
Q_Y                     "Q_F,F_Y"
Q_[]                    "Q_F,F_[@1]"

[]+C_F                  "R SEL/@1,ALU OP/R+C"
[]_F                    "[@1]+C_F,0_CO"
[]_Y                    "[@1]_F,F_Y"
[]_Q                    "[@1]_F,F_Q"
[]_Q RAMFILE_[]         "[@1]_F,F_Q Y_[@2],RAMFILE_Y"
[]_[]                   "[@1]_F,F_[@2]"
[]_Q_[]                 "[@1]_F,F_Q_[@2]"

B[]_F                   "S SEL[@1],0_CO,ALU OP/S+C"

2*Q_Q                   "QLS_Q,0_SIOO 0_QIOO"
2*Q_[]                  "Q_F,LSZ F_[@1]"
2*[]_Q                  "R SEL/@1,S SEL[@1],0_CO,ALU OP/R+S+C,F_Q"
2*[]_[]                 "[@1]_F,LSZ F_[@2]"
2*([]+[])_B             "[@1]+[@2]_F,LSZ F_B"
4*[]_B                  "2*([@1]+[@1])_B"

COMPLEMENT []_F         "R SEL/@1,ALU OP/RBAR+C,0_CO"
COMPLEMENT Q_F          "Q_S,ALU OP/SBAR+C,0_CO"

ASR []_[]               "[@1]_F,B SEL/@2,LRS F_B,MN_SION SIOO_QION"          ;Right shift f,
ASR []_[] MC            "[@1]_F,B SEL/@2,LRS F_B,0_SION SIOO_MC MN_QION"     ;Right shift f,
ASRC []_[]              "[@1]_F,B SEL/@2,ASRC_B,MN_SION SIOO_QION"
ROR []_[]               "[@1]_F,B SEL/@2,LRS F_B,SIOO_SION QIOO_QION"
ROR F_B ROR Q_Q         "LSRC_B,SIOO_SION QIOO_QION"
F LRS_[]                "B SEL/@1,LRS F_B,0_SION 0_QION"
F LRS_[] SIOO_MC        "B SEL/@1,LRS F_B,0_SION SIOO_MC MN_QION"
LRS []_[]               "[@1]_F,F LRS_[@2]"
RORC []_[]              "[@1]_F,B SEL/@2,LSRC_B,QIOO_SIGN SIOO_QION"
LSRC []_[]              "[@1]_F,B SEL/@2,LSRC_B,0_SION SIOO_QION"

ROL []_[]               "[@1]_F,B SEL/@2,ROL F_B"
ROL []_[] ROL Q_Q       "[@1]_F,B SEL/@2,LLSC_B,SION_SIOO QION_QIOO"
LLS []_[]               "[@1]_F,LSZ F_[@2]"
LLS []_[] LLS Q_Q       "[@1]_F,B SEL/@2,LLSC_B,0_SIOO 0_QIOO"
ROLC []_[]              "[@1]_F,B SEL/@2,LLSC_B,SION_QIOO QION_SIOO"
LLSC []_[]              "[@1]_F,B SEL/@2,LLSC_B,QION_SIOO 0_QIOO"

NEGATE Q_F              "Q_S,1_CO,ALU OP/SBAR+C"

[]BAR+C_F               "R SEL/@1,ALU OP/RBAR+C"
NEGATE []_F             "[@1]BAR+C_F,1_CO"
```

```
[]+1_F                  "[@1]+C_F,1_CO"
[]+1_Y                  "[@1]+1_F,F_Y"
[]+1_Q                  "[@1]+1_F,F_Q"
[]+1_Q RAMFILE_[]       "[@1]+1_F,Y_B F_Q,RAMFILE_Y,B SEL/@2"
[]+1_[]                 "[@1]+1_F,F_[@2]"
[]+1_Q_[]               "[@1]+1_F,F_Q_[@2]"
[]+2_B                  "S SEL[@1],1_CO,ALU SP FUN/S+1+C_B,ALU_Y/YES"

Q+[]+C_F                "R SEL/@1,Q_S,ALU OP/R+S+C"
Q+[]+1_F                "Q+[@1]+C_F,1_CO"
Q+[]_F                  "Q+[@1]+C_F,0_CO"
Q+[]_Y                  "Q+[@1]_F,F_Y"
Q+[]_Q                  "Q+[@1]_F,F_Q"
Q+[]_[]                 "Q+[@1]_F,F_[@2]"
[]+[]+C_F               "R SEL/@1,S SEL[@2],ALU OP/R+S+C"
[]+[]+C_B               "[@1]+[@2]+C_F,F_B"
[]+[]_F                 "[@1]+[@2]+C_F,0_CO"
[]+[]_Y                 "[@1]+[@2]_F,F_Y"
[]+[]_Q                 "[@1]+[@2]_F,F_Q"
[]+[]_B                 "[@1]+[@2]_F,F_B"
[]+[]+1_F               "[@1]+[@2]+C_F,1_CO"

Q-[]_F                  "R SEL/@1,Q_S,1_CO,ALU OP/S-R-1+C"
Q-[]_Y                  "Q-[@1]_F,F_Y"
[]-Q_F                  "R SEL/@1,Q_S,1_CO,ALU OP/R-S-1+C"
[]-[]-1+C_F             "R SEL/@1,S SEL[@2],ALU OP/R-S-1+C"
[]-[]-1_F               "[@1]-[@2]-1+C_F,0_CO"
[]-[]_F                 "[@1]-[@2]-1+C_F,1_CO"
[]-[]_Y                 "[@1]-[@2]_F,F_Y"
[]-[]_Q                 "[@1]-[@2]_F,F_Q"
[]-[]_B                 "[@1]-[@2]_F,F_B"
[]-[]-1+C_A             "R SEL/@2,S SEL[@1],ALU OP/S-R-1+C,F_B"
[]-[]-1_A               "[@1]-[@2]-1+C_A,0_CO"
[]-[]_A                 "[@1]-[@2]-1+C_A,1_CO"

SM []_B                 "S SEL[@1],CX_CO,ALU SP FUN/SM,ALU_Y/YES"       ;Sign/magnitude, two's co
MUL [] []               "R SEL/@1,S SEL[@2],0_CO,ALU SP FUN/MUL,0_SION SI00_QION,ALU_Y/YES"
TCM [] []               "R SEL/@1,S SEL[@2],0_CO,ALU SP FUN/TCM,0_SION SI00_QION,ALU_Y/YES"
TCM COR [] []           "R SEL/@1,S SEL[@2],CX_CO,ALU SP FUN/TCM COR,0_SION SI00_QION,ALU_Y/YES"
TCDIV [] []_B           "R SEL/@1,S SEL[@2],CX_CO,ALU SP FUN/TCDIV,SION_QI00 QION_SI00,ALU_Y/YES"
TCDIV COR [] []_B       "R SEL/@1,S SEL[@2],CX_CO,ALU SP FUN/TCDIV COR,1_SI00 1_QI00,ALU_Y/YES"

Q&[]_F                  "R SEL/@1,Q_S,ALU OP/R.AND.S"
Q&[]_Y                  "Q&[@1]_F,F_Y"
Q&[]_Q                  "Q&[@1]_F,F_Q"
Q&[]_Q RAMFILE_[]       "Q&[@1]_F,F_Q Y_[@2],RAMFILE_Y"
Q&[]_[]                 "Q&[@1]_F,F_[@2]"
[]&[]_F                 "R SEL/@1,S SEL[@2],ALU OP/R.AND.S"
[]&[]_Y                 "[@1]&[@2]_F,F_Y"
[]&[]_Q                 "[@1]&[@2]_F,F_Q"
[]&[]_B                 "[@1]&[@2]_F,F_B"

[]BAR&Q_F               "R SEL/@1,Q_S,ALU OP/RBAR.AND.S"
[]BAR&Q_Y               "[@1]BAR&Q_F,F_Y"
[]BAR&[]_F              "R SEL/@1,S SEL[@2],ALU OP/RBAR.AND.S"
[]BAR&[]_Y              "[@1]BAR&[@2]_F,F_Y"
[]BAR&[]_Q              "[@1]BAR&[@2]_F,F_Q"
[]BAR&[]_B              "[@1]BAR&[@2]_F,F_B"
```

```
Q.OR.[]_F               "R SEL/@1,Q_S,ALU OP/R.OR.S"
Q.OR.[]_Y               "Q.OR.[@1]_F,F_Y"
Q.OR.[]_Q               "Q.OR.[@1]_F,F_Q"
Q.OR.[]_[]              "Q.OR.[@1]_F,F_[@2]"
[].OR.[]_F              "R SEL/@1,S SEL[@2],ALU OP/R.OR.S"
[].OR.[]_Y              "[@1].OR.[@2]_F,F_Y"
[].OR.[]_Q              "[@1].OR.[@2]_F,F_Q"
[].OR.[]_B              "[@1].OR.[@2]_F,F_B"
Q.XOR.[]_F              "R SEL/@1,Q_S,ALU OP/R.XOR.S"
[].XOR.[]_F             "R SEL/@1,S SEL[@2],ALU OP/R.XOR.S"
[].XOR.[]_Y             "[@1].XOR.[@2]_F,F_Y"

Q.XNOR.[]_F             "R SEL/@1,Q_S,ALU OP/R.XNOR.S"
[].XNOR.[]_F            "R SEL/@1,S SEL[@2],ALU OP/R.XNOR.S"

READ []                 "[@1]_Q_[E],SPEC SEL/PAGE TABLE ENTRY,CALL [MEMORY READ]"
WRITE []                "[@1]_Q_[E],SPEC SEL/PAGE TABLE ENTRY,CALL [MEM WRITE 1]"

CHECK INTERRUPTS        "IF [PI REQ] CALL [CHK INT]"

SECTION SELECT          "SPEC SEL/PAGE TABLE ENTRY"
;       Things to fix !!!
LIGHTS []       "J[@1]_[PXCT]"
```

```
.TOC    "Dispatch Table Macros"

        .DCODE
J/=<0:11>,.ADDRESS,.DEFAULT=<MCE>       ;1st Dispatch = OPERAND FETCH Dispatch
K/=<12:23>,.DEFAULT=<MCE>       ;2nd Dispatch = INST EXCT Dispatch
L/=<24:35>,.DEFAULT=<MCE>       ;3rd Dispatch = OPERAND STORE Dispatch
M/=<36:47>,.DEFAULT=<MCE>       ;4th Dispatch = final inst dispatch
N/=<48:59>,.DEFAULT=<MCE>       ;5th Dispatch = free
O/=<60:71>,.DEFAULT=<MCE>       ;6th Dispatch = free
P/=<72:83>,.DEFAULT=<MCE>       ;7th Dispatch = 1st half of IO dispatch
Q/=<84:95>,.DEFAULT=<MCE>       ;8th Dispatch = 2nd half of IO dispatch

[] [] [] [] [] [] [] [] "J/@1,K/<J/@2>,L/<J/@3>,M/<J/@4>,N/<J/@5>,O/<J/@6>,P/<J/@7>,Q/<J/@8>"
[]                      "[@1] [MCE] [MCE] [MCE] [MCE] [MCE] [MUUO] [MCE]"
[]#[]                   "[@1] [MCE] [MCE] [MCE] [MCE] [MCE] [@2] [MCE]"
[]##[]                  "[@1] [MCE] [MCE] [MCE] [MCE] [MCE] [MUUO] [@2]"
[] []                   "[@1] [@2] [MCE] [MCE] [MCE] [MCE] [MUUO] [MCE]"
[] []#[]                "[@1] [@2] [MCE] [MCE] [MCE] [MCE] [@3] [MCE]"
[] []##[]               "[@1] [@2] [MCE] [MCE] [MCE] [MCE] [MUUO] [@3]"
[] [] []                "[@1] [@2] [@3] [MCE] [MCE] [MCE] [MUUO] [MCE]"
[] [] []#[]             "[@1] [@2] [@3] [MCE] [MCE] [MCE] [@4] [MCE]"
[] [] []##[]            "[@1] [@2] [@3] [MCE] [MCE] [MCE] [MUUO] [@4]"
[] [] [] []             "[@1] [@2] [@3] [@4] [MCE] [MCE] [MUUO] [MCE]"
[] [] [] []#[]          "[@1] [@2] [@3] [@4] [MCE] [MCE] [@5] [MCE]"
[] [] [] []##[]         "[@1] [@2] [@3] [@4] [MCE] [MCE] [MUUO] [@5]"
        .UCODE
```

```
        .BIN

;First clear the paging ram
BEGIN:  J[6000]_[W1],POP                        ;Will become 30000 next
        ;J[2515]_Y24-Y35,[W1]+[W1]_F,    ;300 baud
        ;J[5115]_Y24-Y35,[W1]+[W1]_F,    ;2400 baud
        J[7115]_Y24-Y35,[W1]+[W1]_F,     ;9600 baud
         F_LSZ Y_[W1],POP                ; Ext clocks, 300 baud,
                                         ; 1 stop bit, 8bits, async 1x
        J[LNOMDW]_RAMFILE ADR,POP        ;CTY mode word
        SWAP [W1]_[W1],
         CALL [W1_RAMFILE]
        ;GOTO [TEST]                     ;Try to verify machine works
                                         ; This also initializes 290's regs
                                         ; and ramfile constants
```

```
;Here to check system seems to be ok
; This clobbers eve ything in Ramfile except CTY status and mode
; After checking will initialize Ramfile and 2903 regs
TEST:
;Initialize some registers
        ONES_[K -1],POP
        J[1]_[W1],POP
        SWAP [W1]_[BIT17],POP

.IF/FTDIAG
;First qki check on 2903, 2904, and 2910
        LIGHTS [1]                          ;1st ALU test
        JN[7776]_[W1]                       ;-2 to W1
        [W1]_Y,
         IF NOT CT [IN] THEN [ALU BROKE]
        [W1]+1_[W1],
         IF CT [IZ] THEN [ALU BROKE]
        [W1]+1_[W1],
         IF NOT CT [IZ] THEN [ALU BROKE]
;       Right shift a bit through Q register
;       Left rotate a bit through W1
;       Count -36 to zero
        JS[0]_[W1]                          ;400000,,0_W1
        [W1]_Q,J[43]_CTR                    ;44(8)=36(10)
        JN[7733]_[W2]                       ;Number to count up to zero
TALU21: Q_F,F_Y LSRQ_Q,                     ;Shift Q right
         IF CT [IZ] THEN [ALU BROKE]
        ROL [W1]_[W1],                      ;Put 1 in W1
         IF CT [IZ] THEN [ALU BROKE]
        [W2]+1_[W2],                        ;Check didn't go to zero yet
         IF CT [IZ] THEN [ALU BROKE]
        LOOP [TALU21]                       ;Back for rest of count
        Q_F,F_Y LSRQ_Q,
         IF NOT CT [IZ] THEN [ALU BROKE]
        ROL [W1]_[W1],                      ;Rotate one more position
         IF NOT CT [IN] THEN [ALU BROKE]
        [K -1]+[W1]_Y,
         IF NOT CT [IZ] THEN [ALU BROKE]
        [W2]+1_[W2],                        ;Final count out
         IF NOT CT [IZ] THEN [ALU BROKE]
```

```
;Verify UX and MX in 2904
; For this test bits 28-31 is UX, 32-35 is MX
        LIGHTS [2]
        J[377]_[W2]                     ;Start with all flags

.IF/FTBB
;Set UX from W2 bits 28-31, set MX from W2 bits 32-35
TUMX12: ROR [W2]_[W1],J[2]_CTR
TUMX13: ROR [W1]_[W1],Y_MX,             ;Load MX from 32-35
         LOOP [TUMX13]
        ROR [W1]_[W1],
         J[2]_CTR,MX_UX                 ;Load UX from 32-35
TUMX14: ROR [W1]_[W1],Y_MX,             ;Load MX from 28-31
         LOOP [TUMX14]
        MX_UX UX_MX                     ;Exchange MX and Ux

;Verify UX & MX by reading onto Y bus
        UX_[W1],J[3]_CTR                ;Get micro flags
TUMX15: ROL [W1]_[W1],LOOP [TUMX15]     ;Put UX in W1 bits 31-35
        J[17]_[W3]                      ;Mask for extra Y bus bits
        [W3]&[W1]_F,F_Q Y_[W1],         ;Mask off extra bits
         MX_Y,J[3]_CTR
TUMX16: ROL [W1]_[W1] ROL Q_Q,          ;Shift left 4 places
         LOOP [TUMX16]
        [W3]&[W1]_B                     ;Mask off extra bits
        Q.OR.[W1]_[W1]
        [W1].XOR.[W2]_Y,
         IF NOT CT [IZ] THEN [ALU BROKE]
.ENDIF/FTBB

.IFNOT/FTBB
;Set UX from W2 bits 28-31, set MX from W2 bits 32-35
TUMX12: LRS [W2]_[W1],J[2]_CTR
TUMX13: LRS [W1]_[W1],Y_MX,             ;Load MX from 28-31
         LOOP [TUMX13]
        [W2]_Y,Y_MX MX_UX               ;Load UX from MX and MX from Y

;Verify UX & MX by reading onto Y bus
        UX_[W1]                         ;Get micro flags
        J[17]_[W3]                      ;Mask for extra Y bus bits
        [W3]&[W1]_B,J[1]_CTR            ;Mask off extra bits
TUMX16: 4*[W1]_B,LOOP [TUMX16]          ;Put UX in W1 bits 28-31
        MX_[W1],[W3]&[W1]_F,F_Q Y_[W1]  ;Mask off extra bits
        Q.OR.[W1]_[W1]                  ;Combine UX and MX
        [W1].XOR.[W2]_Y,
         IF NOT CT [IZ] THEN [ALU BROKE]
.ENDIF/FTBB

;Verify UX & MX with TEST SELECT
        ZERO_[W1],IF CT [UOVR] CALL [W1+1_W1]
        [W1]+[W1]_B,IF CT [UC] CALL [W1+1_W1]
        [W1]+[W1]_B,IF CT [UN] CALL [W1+1_W1]
        [W1]+[W1]_B,IF CT [UZ] CALL [W1+1_W1]
        [W1]+[W1]_B,IF CT [MOVR] CALL [W1+1_W1]
        [W1]+[W1]_B,IF CT [MC] CALL [W1+1_W1]
        [W1]+[W1]_B,IF CT [MN] CALL [W1+1_W1]
        [W1]+[W1]_B,IF CT [MZ] CALL [W1+1_W1]
        [W1].XOR.[W2]_Y,
```

```
 IF NOT CT [IZ] THEN [ALU BROKE]

[K -1]+[W2]_B,
 IF NOT CT [IN] THEN [TUMX12]
```

```
;Verify LOCAL, USER, PAGED, RUN, TOPS20, PXCT, and TRAP latches
        LIGHTS [3]
        J[177]_[W2]                     ;1st check all latches
TLTCH2:
;Here to set LOCAL, USER, PAGED, RUN, TOPS20, PXCT, & TRAP latches from W2
        J[100]_Y24-Y35,[W2]_F,Y_[W3] F_Q
        Q&[W3]_F,F_Y QLS_Q,CLEAR LOCAL,
         IF NOT CT [IZ] CALL [SET LOCAL LATCH]
        Q&[W3]_F,F_Y QLS_Q,CLEAR USER,
         IF NOT CT [IZ] CALL [SET USER LATCH]
        Q&[W3]_F,F_Y QLS_Q,CLEAR PAGED,
         IF NOT CT [IZ] CALL [SET PAGED LATCH]
        Q&[W3]_F,F_Y QLS_Q,CLEAR RUN,
         IF NOT CT [IZ] CALL [SET RUN LATCH]
        Q&[W3]_F,F_Y QLS_Q,CLEAR TOPS20,
         IF NOT CT [IZ] CALL [SET TOPS20 LATCH]
        Q&[W3]_F,F_Y QLS_Q,CLEAR PXCT,
         IF NOT CT [IZ] CALL [SET PXCT LATCH]
        Q&[W3]_F,F_Y QLS_Q,CLEAR TRAP,
         IF NOT CT [IZ] CALL [SET TRAP LATCH]

;Here to read LOCAL, USER, PAGED, RUN, TOPS20, PXCT, & TRAP latches into W1
        ZERO_[W1],IF [LOCAL] CALL [W1+1_W1]
        2*[W1]_[W1],IF [USER] CALL [W1+1_W1]
        2*[W1]_[W1],IF [PAGED] CALL [W1+1_W1]
        2*[W1]_[W1],IF [RUN] CALL [W1+1_W1]
        2*[W1]_[W1],IF [TOPS20] CALL [W1+1_W1]
        2*[W1]_[W1],IF [PXCT] CALL [W1+1_W1]
        2*[W1]_[W1],IF [TRAP] CALL [W1+1_W1]

        CALL [TVR XY]                   ;Be sure W1 and W2 are same

        [K -1]+[W2]_B,                  ;Next combination of latches
         IF NOT CT [IN] THEN [TLTCH2]
```

```
;Verify the PI status register may be read & written
        LIGHTS [4]
        J[7]_[W2],Y_2914 STATUS          ;1st value for reg
        J[7]_[W3]                        ;Mask for value
TPI2:   2914 STATUS_Y,Y_[W1]             ;Get value back from reg
        [W3]&[W1]_B,
         CALL [TVR XY]                   ; Be sure we got correct value
        [K -1]+[W2]_B,Y_2914 STATUS,
         IF NOT CT [IN] THEN [TPI2]

;Verify the PI mask register may be read & written
.IF/FTLDIAG
        LIGHTS [5]
        J[377]_[W2],                     ;1st value for mask
         INT OP/LOAD MASK REG
        J[377]_[W3]                      ;Mask for mask register
TPI3:   Y_[W1],INT OP/READ MASK REG      ;Get value back from mask
        [W3]&[W1]_B,                     ;Mask off extra bits
         CALL [TVR XY]                   ; Be sure we got correct value
        [K -1]+[W2]_B,
         INT OP/LOAD MASK REG,
         IF NOT CT [IN] THEN [TPI3]
         INT OP/LOAD STATUS REG,         ;Next value for register
         IF NOT CT [IN] THEN [TPI2]
.ENDIF/FTLDIAG

;Verify the PI mask register blocks interrupts
        LIGHTS [6]
        J[377]_[W2]                      ;1st value for mask
TPI4:   J[7777]_[W6],                    ;Stall counter
         INT OP/MASTER CLEAR
        [W2]_Y,Y_2914 MASK               ;Inhibit some interrupts
        J[377]_[W1]                      ;Chnls where ints didn't happen
        LRS [W1]_[W3],ALU_PROC REG       ;Start ints on all chnls
TPI7:   J[7]_[W4]                        ;Mask for 2914 status
        [W4]_Q,
         IF [NO PI REQ] THEN [TPI9]
        [W4]+1_Y,ALU_CTR                 ;Load ctr with 10
        IF [TIMER FLAG] THEN [TPI71],
         GRANT INTERRUPT                 ; Grant interrupt
        2914 STATUS_Y,Y_[W4]
        Q&[W4]_Y,ALU_CTR                 ;Get interrupting level+1
TPI71:  BITO_[W4],                       ;Bit for interrupt
         GOTO [TPI73]
TPI72:  ROL [W4]_[W4]
TPI73:  ZERO_Y,Y_2914 STATUS,            ;Allow next interrupt
         LOOP [TPI72]
        [W4]BAR&[W1]_B                   ;Flag we got int
        [W4]BAR&[W3]_B,ALU_PROC REG      ;Stop trying to make that kind
TPI9:   [K -1]+[W6]_B,
         IF NOT CT [IN] THEN [TPI7]
        CALL [TVR XY]                    ;See if right ints were inhibited
        [K -1]+[W2]_B,
         IF NOT CT [IN] THEN [TPI4]
```

```
;Save CTY registers
        J[LNOMDW]_RAMFILE ADR
        RAMFILE_[PC],NRFRD/1            ;Save mode register here

;Ramfile test 1, float a bit through each word in Ramfile
TVR 00: LIGHTS [11]                     ;First ramfile test
        CALL [TVR X0]                   ;Point to highest ramfile adr
TVR 1:  J[1]_[W2]                       ;Bit to float
TVR 2:  [W2]_RAMFILE,IX_MX,             ;Write next pattern in ramfile
         CALL [TVR XX]                  ; Check we could write bit
        2*[W2]_[W2],                    ;Write next word in ramfile
         IF NOT CT [MZ] THEN [TVR 2]
        Q_Y,ALU_RAMFILE                 ;Write words address in word
        Q+[K -1]_F,F_Q_Y,Y_RAMFILE ADR, ;Next ramfile location
         IF NOT CT [IN] THEN [TVR 1]

;Ramfile test 2, verify each word has its address written in it
        LIGHTS [12]                     ;Second ramfile test
        CALL [TVR X0]                   ;Initialize ramfile adr
TVR 3:  Q_[W2],Y_RAMFILE ADR,           ;Address should equal data
         CALL [TVR XX]
        Q+[K -1]_F,F_Q_Y,Y_RAMFILE ADR, ;Next ramfile word
         IF NOT CT [IN] THEN [TVR 3]

;Ramfile test 3, address ramfile with special selects
        LIGHTS [13]                     ;Third ramfile test
        J[1477]_[W2],SET USER           ;Expected page table contents
        J[0000]_[W3]                    ; Illegal section, sect.NE.0,
                                        ; not AC ref, no context match
        ONES_F,F_Y,
         SPEC SEL/PAGE TABLE ENTRY,
         CALL [RDPG LATCHES]
        J[1000]_[W2],SET EXEC           ;Correct page table contents
        J[0016]_[W3]                    ; Legal section, sect.EQ.0
                                        ; AC ref, no context match
        ZERO_Y,ALU_PROC REG,            ;Register block 0
         SPEC SEL/PAGE TABLE ENTRY,
         CALL [RDPG LATCHES]
        J[0701]_[W1]                    ;Will be AC16, XR=1
        SWAP [W1]_[W1]
        J[601]_[W2]
        [W1]_Y,LOAD IR,SPEC SEL/XR,     ;Load IR
         CALL [TVR XX]
        J[600]_[W2]                     ;What should be in AC0 block 0
        ZERO_Y,SPEC SEL/VMA,            ;Address AC0
         CALL [TVR XX]
        SEL AC+[2]                      ;Select AC2 block 0
        CALL [TVR XX]
        J[616]_[W2]
        SEL AC+[0]
        CALL [TVR XX]

;Zero entire Ramfile
        CALL [TVR X0]                   ;Set first adr for ramfile
IRF 2:  ZERO_Y,ALU_RAMFILE
        Q+[K -1]_F,F_Q_Y,Y_RAMFILE ADR,
         IF NOT CT [IN] THEN [IRF 2]
```

```
;Check PCI's are OK
        J[LNOSW]_[W6]                   ;First line block adr

;Verify we can read/write CR register
TPCI0:  LIGHTS [20]
        [W6]_[PXCT LH]
        J[357]_[W4]                     ;Mask for compare
        J[377]_[W3]                     ;Initial value
TPCI1:  J[A1!A0]_[W1]                   ;To address CR register
        [W3]_[W2],CALL [WR PCI]         ;Write CR register
        CALL [RD PCI CR]                ;Get CR now
        [W4]&[W2]_B                     ;Leave only good data
        [W4]&[W1]_B,                    ;Leave only read data
         CALL [TVR XY]                  ; Compare good with bad
        [K -1]+[W3]_B,                  ;Next value for CR
         IF NOT CT [IN] THEN [TPCI1]

;Verify we can read/write MR1/2
        J[21]_[PXCT RH]

        [W6]+1_Y,Y_RAMFILE ADR          ;Address mode word in ramfile
.IF/FTLDIAG
        J[6000]_[W1]                    ;Will become 30000
        J[7777]_Y24-Y35,[W1]+[W1]_F,    ;Leave 37777 in W1
         F_LSZ Y_[W1]
        SWAP [W1]_[W2]                  ;Put 37777,,0 in W2
.ENDIF/FTLDIAG
.IFNOT/FTLDIAG
        J[4000]_[W1]                    ;Will become 020000
        4*[W1]_B                        ;Makes 020000
        SWAP [W1]_[W2]                  ;Make 020000,,0
.ENDIF/FTLDIAG
TPCI2:  [W2]_RAMFILE,
         CALL [SET PCI MODE]
        J[A1]_[W1]                      ;For PCI adr to read MR1/2
        CALL [RD PCI]                   ;Get MR1
        J[377]_[W3]                     ;Mask for extra bits
        [W3]&[W1]_Q                     ;Leave only MR1 in Q
        PCI_ALU,Y_[W1],J[3]_CTR         ;Read MR2
        J[77]_[W2]                      ;Mask for MR25-MR20
        [W2]&[W1]_B
TPCI4:  4*[W1]_B,LOOP [TPCI4]
        Q+[W1]_[W1]                     ;Combine W1 and W2
        [W6]+1_Y,Y_RAMFILE ADR,         ;Address ramfile mode word
         CALL [SWAP W1_W1]              ;Position like ramfile
        RAMFILE_[W2],CALL [TVR XY]      ;Get correct answer
.IF/FTLDIAG
        [W2]-[BIT17]_A,                 ;On to next combination
         IF NOT CT [IN] THEN [TPCI2]
.ENDIF/FTLDIAG
.IFNOT/FTLDIAG
        ROR [W2]_[W2 LH],               ;On to next bit
         IF NOT CT [IZ] THEN [TPCI2]
.ENDIF/FTLDIAG

;Now ready for next line
        J[10]_[W1]                      ;Bit to test
        [BIT17]+[W6]_B,                 ;Next line number in LH
```

```
        CALL [SWAP W1_W1]               ;Position to test line number
       [W1]&[W6]_Y,                     ;See if we are done
        IF CT [IZ] THEN [TPCIO]

Restore CTY Ramfile registers
       J[LNOMDW]_RAMFILE ADR
       [PC]_RAMFILE,
        CALL [CMD&CMR FLUSH]            ;Flush CMD and CMR buffers
```

```
;Checksum DISPATCH RAM
        LIGHTS [40]                         ;Check DISPATCH ROM
        ZERO_F,F_Q_[W1]                     ;Initialize checksum
        J[0777]_[W2]                        ;First IR
TDP1:   SWAP [W2]_[W3],J[3]_CTR
TDP2:   4*[W3]_B,LOOP [TDP2]
        2*[W3]_[W3],LOAD IR
        DISP [10]_Y,Q+[W1]_F,F_Q Y_[W1]
        DISP [11]_Y,Q+[W1]_F,F_Q Y_[W1]
        DISP [12]_Y,Q+[W1]_F,F_Q Y_[W1]
        DISP [13]_Y,Q+[W1]_F,F_Q Y_[W1]
        DISP [14]_Y,Q+[W1]_F,F_Q Y_[W1]
        DISP [15]_Y,Q+[W1]_F,F_Q Y_[W1]
        DISP [16]_Y,Q+[W1]_F,F_Q Y_[W1]
        DISP [17]_Y,Q+[W1]_F,F_Q Y_[W1]
        [K -1]+[W2]_B,                      ;Next value for IR
         IF NOT CT [IN] THEN [TDP1]
        Q+[W1]_[W1]                         ;Make final checksum
        J[7777]_[W3]                        ;Mask for checksum
        J[0105]_[W2]                        ;Known checksum
        [W3]&[W1]_B,                        ;Mask result to 12 bits
         CALL [TVR XY]                      ; Compare two values

;Check memory is OK
        ;GOTO [SYS INIT]                    ;Initialize everything
.ENDIF/FTDIAG
.IFNOT/FTDIAG
        J[LNOMDW]_RAMFILE ADR               ;Address Cty mode word
        J[2000]_[W1]
        RAMFILE_[W6]                        ;Save it
        ZERO_[W2],Y_RAMFILE ADR
IR2:    ZERO_Y,ALU_RAMFILE,
         CALL [W2+1_RFA]
        [K -1]+[W1]_B,
         IF NOT CT [IZ] THEN [IR2]
        J[LNOMDW]_RAMFILE ADR               ;Address Cty mode word
        [W6]_Y,ALU_RAMFILE,
         CALL [CMD&CMR FLUSH]               ;Flush CMD and CMR buffers
.ENDIF/FTDIAG
```

```
;Here after TEST has run, to initialize machine for running
SYS INIT:

;Setup 2903 registers
        J[0077]_[K 77]                  ;Initialize register
        J[777]_[K 777],                 ;Initialize register
         CLEAR RUN
        J[27.]_[TIME]                   ;Initialize register
        ZERO_[K -1.0 RH]                ;Clear RH of register
        J[3]_[CO!C1]
        ONES_[K -1.0 LH],PUSH J[2]_CTR  ;Set LH of register
        ROR [CO!C1]_[CO!C1],RFCT        ;Load 300000,,0
        J[1]_[W1]
        SWAP [W1]_[BIT17]               ;Makes 1,,0
        J[7777]_Y24-Y35,[BIT17]_F,      ;Makes 0,,4077777
         F_ROR Y_[W1]
        SWAP [W1]_[K 407777.0]          ;Initialize register
        J[0040]_[W1]
        SWAP [W1]_[BIT12]

;Setup RAMFILE byte masks
        J[BYTE MASK]_[W1]               ;First adr in byte mask area
        [W1]+[K -1]_Q,                  ;First adr in Ramfile to set
         J[77]_CTR                      ;Number of masks to write
        ZERO_[W1]                       ;First word is all zeros
IRF 10: CALL [RF WRITE]
        [W1]_F,LLS F_B,1_S100 1_Q100,   ;Shift left shifting in ones
         B SEL/W1,LOOP [IRF 10]

;Setup RAMFILE bit table
        BIT0_[W1]                       ;First bit
IRF 12: CALL [RF WRITE]
        LRS [W1]_[W1],
         IF NOT CT [IZ] THEN [IRF 12]

;Setup other Ramfile locations
        J[PUSHJ FLAGS]_RAMFILE ADR
        J[4340]_[W1]                    ;Want 021600,,0
        SWAP [W1]_[W1]
        4*[W1]_B,ALU_RAMFILE,
         J[5]_CTR

;Setup M8628 status register
        J[BIT20]_RAMFILE ADR            ;Want 100000
        J[7]_[W5]                       ;Highest possible memory
        RAMFILE_[W4],J[1]_CTR
        J[6540]_[W1]                    ;Clear error flags
IM 12:  4*[W1]_B,LOOP [IM 12]
IM 20:  [W4]+[W5]_F,F_Q_Y,              ;Address memory status reg
         START IO WRITE
        [W1]_[W1],ALU_IO                ;Write status register
        [K -1]+[W5]_B,                  ;Next memory
         IF NOT CT [IN] THEN [IM 20]    ; Loop back for rest of memories

;Setup another 2903 reg
        J[BIT4]_RAMFILE ADR
        RAMFILE_[BIT4],NRFRD/1
        J[30B MASK]_RAMFILE ADR
```

```
        RAMFILE_[K 7777.-1],NRFRD/1,
         CALL [SYS RESET]                ;Reset a bunch of things

CONSOLE PROMPT:
        CALL [PUT CMR PROMPT]            ;Type prompt
        ;GOTOP [CONSOLE]
```

```
.TOC    "CONSOLE PROGRAM"

CONSOLE:
        IF [RUN] THENP [CSL2]
        CHECK INTERRUPTS
        J[INT RCOVR]_RAMFILE ADR        ;Address recovery routine
        RAMFILE_[W1],NRFRD/1
        [W1]_[W1],
         IF NOT CT [IZ] THEN Y
CSL2:
CMD PRSEX:
        J[CMDTKR]_RAMFILE ADR
.IF/FTCRAM
        RAMFILE_[W1],NRFRD/1            ;Get break char flag
        [W1]_Y,
         IF CT [IZ] THEN [CSL3]
        J[CMRTKR]_RAMFILE ADR           ;Address response putter
        RAMFILE_[W1],NRFRD/1            ;Get response taker
        [W1]_Y,
         IF CT [IZ] THEN [CMD PRSE]
.ENDIF/FTCRAM
.IFNOT/FTCRAM
        RAMFILE_[W1],NRFRD/1,           ;Get break char flag
         IF CT [IZ] THEN [CSL3]
        J[CMRTKR]_RAMFILE ADR           ;Address response putter
        RAMFILE_[W2],NRFRD/1,           ;Get response taker
         IF CT [IZ] THEN [CMD PRSE]
.ENDIF/FTCRAM
CSL3:   IF [RUN] THENP [IFETCH]
.IF/SH&TL
        CALL [SHOW&TELL]
.ENDIF/SH&TL
        GOTOP [CONSOLE]
.IF/SH&TL
SHOW&TELL:
        J[RF PROC REG]_RAMFILE ADR      ;Address ramfile copy of
        RAMFILE_[W1],NRFRD/1            ; PROC REG
        2914 MASK_Y,Y_[W1]
        2914 STATUS_Y,Y_[W1]
        J[LNOSW]_[W6]
        CALL [RD PCI CR]
        [W1]_[W1],
         CALL [RD PCI SR]
        [W1]_[W1]                       ;Display SR
        J[A1]_[W1]                      ;For PCI adr to read MR1/2
        CALL [RD PCI]                   ;Get MR1
        [W1]_[W1]                       ;Display MR1
        PCI_ALU,Y_[W1],                 ;Read MR2
         RETURN
.ENDIF/SH&TL
```

```
;Here to decode next console command
CMD PRSE:
        ZERO_[W5],                      ;Build command in W5
         CALL [GET CMD CHX]             ;Get lst nonblank char from command
        J[15]_[W2]                      ;Ascii <cr>"
        [W1]-[W2]_Y,                    ;If end of line done
         IF CT [IZ] THEN [CMD DONE]
        J[54]_[W2]                      ;Ascii ","
        [W1]-[W2]_Y,
         IF CT [IZ] THEN [CMD PRSE8]    ;Skip command separator
        GOTO [CMDP5]

CMDP2:  CALL [GET CMD CHR]              ;Get next command char
        J[40]_[W2]                      ;Ascii " "
        [W2]-[W1]_Y,                    ;Compare
         IF CT [IZ] THEN [CMDDSP]
        J[15]_[W2]                      ;Ascii "<cr>"
        [W2]-[W1]_Y,                    ;Compare
         IF CT [IZ] THEN [CMDDSP]
        J[54]_[W2]                      ;Ascii ","
        [W2]-[W1]_Y,
         IF CT [IZ] THEN [CMDDSP]
        2*[W5]_[W5],
         J[2]_CTR
CMDP3:  4*[W5]_B,
         LOOP [CMDP3]
CMDP5:  J[140]_[W2]                     ;141= lower case "a"
        [W1]-[W2]_Y,
         IF CT [IN] THEN [CMDP6]
        J[173]_[W2]                     ;172 = lower case "z"
        [W1]-[W2]_Y,
         IF NOT CT [IN] THEN [CMDP6]
        J[40]_[W2]                      ;Mask for bit
        [W2].XOR.[W1]_F,F_B             ;Convert to upper case
CMDP6:  [W1]+[W5]_B,
         GOTO [CMDP2]
```

```
;Consol commands include:

        B       ;(102) Bootstrap
;       C       ;(103) Continue
;       D #1 #2 ;(104) Deposit #2 in physical memory location #1
;       E #     ;(105) Examine physical memory location #
;       H       ;(110) Halts the machine and prints out PC
;       I       ;(111) Initialize system
;       R       ;(122) Repeat
;       S #     ;(123) Start at address #
;       T       ;(124) Perform self test
;       ^C      ;(003) Abort repeat, or current line
;       <cr>    ;(015) ends command
;       ^U      ;(025) deletes current command line
;       ^\      ;(034) Toggles consol/user switch for consol tty
;       ;       ;Separate commands
;       rubout  ;(177) deletes previous typed character

CMDDSP: CALL [BKUP CMD TKR]             ;Backup over command terminator
        J[CMDERR MER]_[W1],POP          ;Where to go if memory fails
        CALL [SET PF ME RCOVR]          ;In case error in command
        J[102]_[W1],POP                 ;Ascii "B"
        [W1]-[W5]_F,F_Q_Y,
         IF CT [IZ] THEN [B CMD]
.IF/FTADRB
        Q+[K -1]_F,F_Y,                 ;Ascii "A" = 101
         IF CT [IZ] THEN [A CMD]
 ENDIF/FTADRB
        Q+1_F,F_Q_Y,                    ;Ascii "C" = 103
         IF CT [IZ] THEN [C CMD]
        Q+1_F,F_Q_Y,                    ;Ascii "D" = 104
         IF CT [IZ] THEN [D CMD]
        Q+1_F,F_Q_Y,                    ;Ascii "E" = 105
         IF CT [IZ] THEN [E CMD]
        J[110]_[W1]                     ;Ascii "H"
        [W1]-[W5]_F,F_Q_Y,
         IF CT [IZ] THEN [H CMD]
        Q+1_F,F_Q_[W1],                 ;Ascii "I" = 111
         IF CT [IZ] THEN [I CMD]
.IF/FTFCSL
        [W1]+2_B,                       ;Ascii "K" 113
         IF CT [IZ] THEN [JK CMD]
.ENDIF/FTFCSL
        J[122]_[W1]                     ;Ascii "R"
        [W1]-[W5]_F,F_Q_Y,
         IF CT [IZ] THEN [REPEAT]
        Q+1_F,F_Q_Y,                    ;Ascii "S" = 123
         IF CT [IZ] THEN [S CMD]
        Q+1_F,F_Q_Y,                    ;Ascii "T" = 124
         IF CT [IZ] THEN [T CMD]
        J[4000]_[W1]
        J[4710]_Y24-Y35,                ;Ascii "SH" = 24710
         [W1]+[W1]_F,F_LSZ Y_[W1]
        [W1]-[W5]_F,F_Q_Y,
         IF CT [IZ] THEN [SH CMD]
        Q+1_F,F_Q_Y,                    ;Ascii "SI" = 24711
         IF CT [IZ] THEN [SI CMD]
.IF/FTFCSL
```

```
        [W1]J[1111]_[W1]                ;Ascii "DI" = 21111
        [W1]-[W5]_Y,
         IF CT [IZ] THEN [DI CMD]
        [W1]J[1122]_[W1]                ;Ascii "DR" = 21122
        [W1]-[W5]_Y,
         IF CT [IZ] THEN [DR CMD]
        [W1]J[1126]_[W1]                ;Ascii "DV" = 21126
        [W1]-[W5]_Y,
         IF CT [IZ] THEN [DV CMD]
        [W1]J[1311]_[W1]                ;Ascii "EI" = 21311
        [W1]-[W5]_Y,
         IF CT [IZ] THEN [EI CMD]
        [W1]J[1326]_[W1]                ;Ascii "EV" = 21326
        [W1]-[W5]_Y,
         IF CT [IZ] THEN [EV CMD]
        [W1]J[1322]_[W1]                ;Ascii "ER" = 21322
        [W1]-[W5]_Y,
         IF CT [IZ] THEN [ER CMD]
.IF/FTSM
        [W1]J[4715]_[W1]                ;Ascii "SM" = 24715
        [W1]-[W5]_Y,
         IF CT [IZ] THEN [SM CMD]
.ENDIF/FTSM
.IF/FTZM
        [W1]J[6515]_[W1]                ;Ascii "ZM" = 26515
        [W1]-[W5]_Y,
         IF CT [IZ] THEN [ZM CMD]
.ENDIF/FTZM
.ENDIF/FTFCSL
;Here if user typed an invalid command
CMDERR URC:
        CALL [CMDBUF FLUSH]             ;Done with CMD buffer
        J[ERR.URC]_[W1]                 ;Unrecognized command
        GOTOP [CMDERR]

CMDERR NWR:
        J[ERR.NWR]_[W1]                 ;Error code (not while running)
        GOTOP [CMDERR]

CMDERR NXA:
        J[ERR.NXA]_[W1]                 ;Error code (nonexistent adr)
        GOTOP [CMDERR]

CMDERR E MER:
        CALL [PUT CMR NUM]              ;Type error status
        CALL [PUT CMR SPACE]
        [MB]_[W1],                      ;Type data next
         GOTO [CMDERR MER]
CMDERR MER:
        CALL [PUT CMR NUM]              ;Type error status
        J[ERR.MER]_[W1]                 ;Error code (memory error)
        ;GOTOP [CMDERR]

CMDERR: J[103]_[W6]                     ;Ascii "C"
        [W1]_[W5],                      ;Save error code
         CALL [PUT ? W6]
        SWAP [W5]_[W1],
         CALL [P6DNUM]                  ;Type (6 digit) error number
        GOTOP [CMD DONE 2]
```

```
CMD DONE:
        CALL [CMDBUF FLUSH]             ;Done with CMD buffer
CMD DONE 2:
        J[CMDFLG]_RAMFILE ADR,POP       ;Addess command flags
        RAMFILE_[W1],NRFRD/1,POP        ;Get flags
        [BIT4]&[W1]_Y,                  ;(F.USR) Check user mode
         IF CT [IZ] CALL [PUT CMR PROMPT]
CMD9:

;Here when command has been executed
CMD PRSE8:
        ZERO_[W1],CALL [SET INT RCOVR]  ;Clear interrupt recovery
        ZERO_[W1],                      ;Clear page fail
         CALL [SET PF ME RCOVR] ; and mem err recovery
        GOTOP [CONSOLE]
```

```
;Here to get a char from a command line, skiping leading blanks
; Returns with character in W1
GET CMD CHX:
        CALL [GET CMD CHR]
        J[40]_[W2]
        [W1]-[W2]_Y,
         IF CT [IZ] THEN [GET CMD CHX]
        RETURN

;Here to get a number from a command line
; Returns clears UZ if number is found, number in W1
GET CMD NUM:
        ZERO_[W5],1_UZ,                 ;Build number here
         CALL [GET CMD CHX]             ; Get 1st nonblank char from command
GCN 2:  J[67]_[W2]
        [W2]-[W1]_Y,
         IF CT [IN] THEN [GCN 4]
        J[60]_[W2]
        [W1]-[W2]_A,
         IF CT [IN] THEN [GCN 4]
        4*[W5]_B,0_UZ                   ;Clear UZ (we got a num)
        2*[W5]_[W5]
        [W1]+[W5]_B,
         CALL [GET CMD CHR]
        GOTO [GCN 2]

GCN 4:  CALL [BKUP CMD TKR]             ;Backup command buffer taker
        [W5]_[W1],                      ;Copy result
         RETURN                         ; and dismiss
```

```
;Here to type number on CTY
PUT CMR NUM:
        CALL [P6DNUM]                   ;Type first half
        J[54]_[W1]                      ;Ascii ","
        CALL [PUT CMR CHR]
        J[X1]_RAMFILE ADR
        RAMFILE_[W1],NRFRD/1,           ;Get rest of number back
         GOTO [P6DNUM]                  ;Type 2nd half

;Here to type a six digit octal number on CTY
P6DNUM: J[6]_[W6]                       ;Keep digit counter in W6
        J[X1]_RAMFILE ADR
P6DNM2: J[2]_CTR
P6DNM3: ROL [W1]_[W1],ALU_RAMFILE,      ;Rotate left thrice
         LOOP [P6DNM3]
        J[7]_[W2]                       ;Mask for number
        [W2]&[W1]_F,F_Q FJ[060]_[W1]    ;Mask number to 3 bits
        Q+[W1]_[W1],                    ;Convert octal to ascii
         CALL [PUT CMR CHR]             ; and print
        J[X1]_RAMFILE ADR               ; and address word
        [K -1]+[W6]_B,IX_MX             ;Count digits typed
        RAMFILE_[W1],                   ;Get number back again
         IF NOT CT [MZ] THEN [P6DNM2]
        RETURN

;Here to type "^x"
; call with J[x]_[W6]
PUT CNTRL W6:
        J[136]_[W1]                     ;Ascii "^"
PUTXW6: CALL [PUT CMR CHR]
        [W6]_[W1],
         GOTO [PUT CMR CHR]

;Here to type "?x"
; Call with J[x]_[W6]
PUT ? W6:
        CALL [PUT CMR CRLF]             ;Begin with a flourish
        J[034]_[W1]                     ;Ascii "^\"
        CALL [PUT CMR CHR]              ;Type it for John Kirchoff
        J[77]_[W1]                      ;Ascii "?"
        GOTO [PUTXW6]

;Here to type a "/" followed by a " "
PUT CMR SLSH:
        J[57]_[W1]                      ;Ascii "/"
        CALL [PUT CMR CHR]
PUT CMR SPACE:
        J[040]_[W1]                     ;Ascii " "
        GOTO [PUT CMR CHR]

;Here to type "<CR><LF>KT20> "
PUT CMR PROMPT:
        CALL [PUT CMR CRLF]             ;Begin with a flourish
        J[CMDTKR]_RAMFILE ADR           ;Address command taker
.IF/FTCRAM
        RAMFILE_[W1],NRFRD/1            ;If already have command
        [W1]_[W1],                      ;Let latches go
```

```
         IF NOT CT [IZ] RETURN          ; don't prompt
.ENDIF/FTCRAM
.IFNOT/FTCRAM
        RAMFILE_[W1],NRFRD/1,           ;If already have command
         IF NOT CT [IZ] RETURN          ; don't prompt
.ENDIF/FTCRAM
        J[113]_[W1]                     ;Ascii "K"
        CALL [PUT CMR CHR]
        J[124]_[W1]                     ;Ascii "T"
        CALL [PUT CMR CHR]
        J[62]_[W1]                      ;Ascii "2"
        CALL [PUT CMR CHR]
        J[60]_[W1]                      ;Ascii "0"
        CALL [PUT CMR CHR]
        J[76]_[W1]                      ;Ascii ">"
        GOTO [PUT CMR CHR]

;Here to type a <CR><LF>
PUT CMR CRLF:
        J[15]_[W1]
        CALL [PUT CMR CHR]
        J[12]_[W1]
        ;CALL [PUT CMR CHR]
        ;RETURN

;Here to type command responses on CTY
PUT CMR CHR:
        J[CMDFLG]_RAMFILE ADR           ;Address command flags
        J[F.0]_[W2]                     ;Flag for we are flushing output
        RAMFILE_Y,[W2]_F,Y_[W2] F_Q     ;Get flags
        Q&[W2]_Y,IF NOT CT [IZ] RETURN  ;If flushing we are done
        J[CMRPTR]_RAMFILE ADR           ;Address putter for CMRBUF
        J[CMRBUF END]_[W2]              ;Last location in CMRBUF
        RAMFILE_[W3]                    ;Get putter
        [W2]-[W3]_Y,                    ;If full can't put in
         IF CT [IZ] RETURN
        CALL [CMB NXT]                  ;Adjust pointer
        [W2]_RAMFILE                    ;Write byte into ramfile
        J[CMRTKR]_RAMFILE ADR           ;Address taker
.IF/FTCRAM
        RAMFILE_[W1],NRFRD/1            ;Get taker
        [W1]_[W1],                      ;Let latches go
         IF CT [IZ] CALL [CMR PTR RESET]
.ENDIF/FTCRAM
.IFNOT/FTCRAM
        RAMFILE_[W1],NRFRD/1,           ;Get taker
         IF CT [IZ] CALL [CMR PTR RESET]
.ENDIF/FTCRAM
        ZERO_[W6 LH],                   ;Cty is line 0
         GOTO [SET PCI TXEN]            ;Start transmitter
```

```
;Here to put characters in the command buffer
; With 7bit char in Q
CONSOLE CHAR:
        Q_[W1],                         ; Flush nulls
         IF CT [IZ] THENP [PCI INT CLR]
        J[003]_[W2]                     ;Ascii "^C"
        Q-[W2]_Y,
         IF CT [IZ] THEN [CNTRL C U]
        J[017]_[W2]                     ;Ascii "^O" = 17
        [W2]-[W1]_A,
         IF CT [IZ] THEN [CNTRL O]
        [W2]+2_B,                       ;Ascii "^Q" = 21 = Xon
         IF CT [IZ] THEN [CNTRL Q]
        [W2]+2_B,                       ;Ascii "^S" = 23 = Xoff
         IF CT [IZ] THEN [CNTRL S]
        J[CMDTKR]_RAMFILE ADR           ;Address command taker
.IF/FTCRAM
        RAMFILE_[W3],NRFRD/1            ;Get taker
        [W3]_[W3],                      ;Let latches go
         IF NOT CT [IZ] THEN [TYPE AHEAD]       ; Ignore type ahead
.ENDIF/FTCRAM
.IFNOT/FTCRAM
        RAMFILE_[W3],NRFRD/1,           ;Get taker
         IF NOT CT [IZ] THEN [TYPE AHEAD]       ; Ignore type ahead
.ENDIF/FTCRAM
.IF/FTCTLR
        [W2]+[K -1]_Y,                  ;Ascii "^R" = 22
         IF CT [IZ] THEN [CNTRL R]
.ENDIF/FTCTLR
        [W2]+2_B,                       ;Ascii "^U" =25
         IF CT [IZ] THEN [CNTRL C U]
        J[032]_[W2]                     ;Ascii "^Z"
        Q-[W2]_Y,
         IF CT [IZ] THEN [CNTRL Z]
        J[177]_[W2]                     ;Ascii "rubout" = 177
        Q-[W2]_Y,                       ;Check for character was a rubout
         IF CT [IZ] THEN [CMD RUBOUT]
        J[CMDPTR]_RAMFILE ADR           ;Address CMDBUF putter
        J[CMDBUF END]_[W2]              ;Last location in CMDBUF
        RAMFILE_[W3]                    ;Get putter
        [W2]-[W3]_Y,                    ;If buffer is full stop
         IF CT [IZ] THEN [REFUSE CHAR]  ; accepting chars
        J[15]_[W2]                      ;Carriage return
        [W1]-[W2]_Y,
         IF CT [IZ] THEN [CMDC 2]
        J[12]_[W6]                      ;Ascii <LF>
        [W6]-[W1]_Y,
         IF NOT CT [IZ] THEN [CMDC 3]

;Here when char is <CR> or <LF>
CMDC 2: [W2]_[W1],                      ;Put <CR> in CMD buffer
         CALL [PUT CMDBUF]
        CALL [ENABLE PARSE]
        CALL [PUT CMR CRLF]             ;End echo with <CR><LF>
CMDC X: IF [NOT RUN] THENP [PCI INT CLR]
.IF/DEBUG3
        J[POOP FLAG]_RAMFILE ADR
        ZERO_Y,ALU_RAMFILE
```

```
.ENDIF/DEBUG3
        ZERO_[W1],CALL [SET PF RCOVR]
        [PXCT]_[PI REG],Y_2914 STATUS,  ;Restore previous level
         GOTO [CMD PRSEX]               ;This isn't really an int

ENABLE PARSE:
        J[CMDTKR]_RAMFILE ADR           ;Point to command taker
        GOTO [CMD PTR RESET]            ;We can parse now

CMDC 3: [W1]_[W6],                      ;Save character
         CALL [PUT CMDBUF]              ; And put in CMDBUF
        [W6]_[W1],                      ;Then echo the char
         GOTOP [PUT W1&DISMISS]         ;Type W1 then dismiss int
```

```
;Here for typeahead
TYPE AHEAD:
        J[CMDRPT]_RAMFILE ADR           ;Address repeat counter
        ONES_F,F_Y,ALU_RAMFILE          ;typeahead kills repeat
REFUSE CHAR:
        J[007]_[W1]                     ;Ascii "^G"
PUT W1&DISMISS:
        CALL [PUT CMR CHR]
        GOTOP [PCI INT CLR]

;Here to process a rubout
CMD RUBOUT:
        J[CMDPTR]_RAMFILE ADR,          ;Address putter
         [K 777]_Q
        RAMFILE_[W1],NRFRD/1            ;Get command putter
        Q&[W1]_F,                       ;Leave only ramfile adr in Q
         F_Q FJ[CMDBUF-1]_[W2]
        [W2]-[W1]_Y,                    ;See if any characters to delete
         IF CT [IZ] THENP [PCI INT CLR] ; Nothing doesn't echo
        CALL [BKUP POINTER]             ;Backup pointer
        J[134]_[W1]                     ;Ascii "\"
        GOTOP [PUT W1&DISMISS]          ;Type W1 then dismiss int

.IF/FTCTLR
;Here for a ^R
CNTRL R:
        J[CMRTKR]_RAMFILE ADR           ;Address response taker
 IF/FTCRAM
        RAMFILE_[W2],NRFRD/1            ;When typing ignore
        [W2]_[W2],                      ;Let latches go
         IF NOT CT [IZ] THENP [PCI INT CLR]
.ENDIF/FTCRAM
.IFNOT/FTCRAM
        RAMFILE_[W2],NRFRD/1,           ;When typing ignore
         IF NOT CT [IZ] THENP [PCI INT CLR]
.ENDIF/FTCRAM
        J[122]_[W6]                     ;Put "R" in W6
        CALL [PUT CNTRL W6]             ;Type ^R
        CALL [PUT CMR PROMPT]           ;Retype prompt
;Copy CMD buffer to CMR buffer
        CALL [ENABLE PARSE]
        GOTO [CR4]                      ;Be sure there is anything to do
CR2:    CALL [GET CMD CHR]
        CALL [PUT CMR CHR]              ;Copy byte to CMR buffer
CR4:    J[CMDPTR]_RAMFILE ADR           ;Address putter
        RAMFILE_[W2],NRFRD/1
        J[CMDTKR]_RAMFILE ADR           ;Address taker
        RAMFILE_[W1],NRFRD/1
        [W1].XOR.[W2]_Y,
         IF NOT CT [IZ] THEN [CR2]
        ZERO_Y,ALU_RAMFILE,             ;Reset CMD taker
         GOTOP [PCI INT CLR]
.ENDIF/FTCTLR
```

```
;Here for a ^\
CNTRL BKSLSH:
;Here for ^C or ^U
CNTRL C U:
        J[!00]_[W6]                     ;To convert "^C" to "C"
        Q+[W6]_[W6],
         CALL [CMD&CMR FLUSH]           ;Flush command buffer
        [BIT4]_F,FJ[F.O!F.S]_[W1]       ;(F.USR) Flags to clear
        CALL [CLEAR CMDFLG]
        CALL [PUT CNTRL W6]             ;Type ^C or ^U
        CALL [PUT CMR PROMPT]
        ZERO_[W1],CALL [SET INT RCOVR]  ;Clear interrupt recovery
        GOTOP [PCI INT CLR]             ;Dismiss interrupt

;Here to turn off F.O (the control O flag)
F.O CLEAR:
        J[F.O]_[W1]                     ;Flags to clear for new command
        GOTO [CLEAR CMDFLG]

;Here for ^O
CNTRL O:
        CALL [CMRBUF FLUSH]             ;Flush command response buffer
        J[117]_[W6]                     ;Ascii "O"
        CALL [PUT CNTRL W6]             ;Type ^O
        J[CMDFLG]_RAMFILE ADR           ;Address ramfile word
        J[F.O]_[W1]                     ;Flag we are flushing output
        RAMFILE_Y,[W1]_F,Y_[W1] F_Q     ;Get word from ramfile
        Q.XOR.[W1]_F,F_Y,ALU_RAMFILE,   ;Set/clear flag
         GOTOP [PCI INT CLR]            ;Dismiss interrupt
```

```
;Here to clear a flag in the CMDFLG word
CLEAR CMDFLG:
        J[CMDFLG]_RAMFILE ADR           ;Address ramfile word
        RAMFILE_Y,NRFRD/1,Y_[W1] F_Q,   ;Get old flag word
         COMPLEMENT [W1]_F
        Q&[W1]_Y,ALU_RAMFILE,           ;Clear flag
         RETURN

;Here for a ^S (Xoff) command
CNTRL S:
        J[F.S]_[W1]                     ;Flag to set
        CALL [SET CMDFLG]               ;Set it
        GOTOP [PCI INT CLR]

;Here to set a flag in the CMDFLG word
SET CMDFLG:
        J[CMDFLG]_RAMFILE ADR           ;Address ramfile word
        RAMFILE_Y,NRFRD/1,[W1]_F,Y_[W1] F_Q     ;Get old flag word
        Q.OR.[W1]_Y,ALU_RAMFILE,        ;Set flag
         RETURN
```

```
;Here for a ^Z
CNTRL Z:
        IF [NOT RUN] THENP [PCI INT CLR]        ;No user mode if stopped
        J[132]_[W6]                             ;Ascii "Z"
        CALL [PUT CNTRL W6]                     ;^Z
        [BIT4]_[W1],                            ;(F.USR) User mode
         CALL [SET CMDFLG]                      ; Set flag
        ;GOTO [CNTRL Q]                         ;Also clear XOF flag

;Here for a ^Q (Xon)
CNTRL Q:
        J[F.S]_[W1]                             ;Flag to clear
        CALL [CLEAR CMDFLG]
        CALL [SET PCI TXEN]                     ;Start transmitter again
        GOTOP [PCI INT CLR]                     ;Dismiss interrupt
```

```
;Here to flush command and command response buffer
CMD&CMR FLUSH:
        CALL [CMRBUF FLUSH]             ;First flush the response buffer
        ;GOTO [CMDBUF FLUSH]

;Here to flush command buffer
CMDBUF FLUSH:
        J[CMDTKR]_RAMFILE ADR
        ZERO_Y,ALU_RAMFILE              ;Clear taker
        J[CMDRPT]_RAMFILE ADR           ;Address repeat counter
        ZERO_Y,ALU_RAMFILE              ;Clear repeat counter
        J[CMDPTR]_RAMFILE ADR
CMD PTR RESET:
        J[CMDBUF-1]_[W1]                ;Adr of 1st word-1 of buffer
        [W1]_RAMFILE,                   ;Initialize putter
         RETURN

;Here to flush command response buffer
CMRBUF FLUSH:
        J[CMRTKR]_RAMFILE ADR
        ZERO_Y,ALU_RAMFILE              ;Clear taker
        J[CMRPTR]_RAMFILE ADR
CMR PTR RESET:
        J[CMRBUF-1]_[W1]                ;Adr of 1st word-1 of buffer
        [W1]_RAMFILE,                   ;Initialize putter
         RETURN
```

```
;Routine to put a byte into the ramfile

; Call with     [W1] = char to put
;       Will first increment pointer than store indirect it
PUT CMDBUF:
        J[CMDPTR]_RAMFILE ADR           ;Address putter for command buffer
PUT CMB:
        CALL [CMB NXT]                  ;Put byte in ramfile word
        [W2]_RAMFILE,                   ;Put word back in ramfile
         RETURN

;Here to get a char from a command line
GET CMD CHR:
        J[CMDTKR]_RAMFILE ADR           ;Address command char taker
        ;GOTO [CMB NXT]                 ;Advance to next byte

;Get or prepare to put next byte in ramfile CMD or CMR buffer
; Call  J[taker or putter]_RAMFILE ADR
;       _[W1]   ;Byte to be stored
; returns
;       [W1]    ;byte from ramfile
;       [W2]    ;Ramfile word with byte inserted
CMB NXT:
        RAMFILE_[W2]                    ;Get current pointer
        [K -1.0]+[W2]_B,                ;Next byte adr
         IF NOT CT [IN] THEN [CMBP4]
        J[4]_[W3]                       ;First byte is number 4
        SWAP [W3]_[W2 LH]               ;Put byte number in LH
        [W2]+1_[W2 RH]                  ;Next ramfile adr
CMBP4:  [W2]_RAMFILE,                   ;Store updated pointer
         Y_RAMFILE ADR                  ; and address buffer
        J[177]_[W3]                     ;Get mask for data
        [W3]&[W1]_F,Y_[W1] F_Q,
         RAMFILE_Y,                     ;Get word from ramfile
         GOTO [CMBP7]
CMBP5:  LRS [W1]_[W1]                   ;Shift ramfile word right
        LLSC [W3]_[W3],                 ;Shift mask and byte left
         J[5]_CTR
CMBP6:  LRS [W1]_[W1]                   ;Shift ramfile word right
        LLSC [W3]_[W3],                 ;Shift mask and byte left
         LOOP [CMBP6]
CMBP7:  [K -1.0]+[W2]_B,
         IF NOT CT [IN] THEN [CMBP5]
        J[177]_[W2]                     ;Get mask for data
        [W2]&[W1]_B                     ;Leave only data byte
        RAMFILE_[W2]                    ;Get ramfile word again
        [W3]BAR&[W2]_B                  ;Clear slot for byte
        Q.OR.[W2]_[W2],                 ;Add new byte
         RETURN
```

```
;Here to backup the command buffer taker
BKUP CMD TKR:
        J[CMDTKR]_RAMFILE ADR
        ;GOTO [BKUP POINTER]

;Here to backup a pointer
BKUP POINTER:
        J[4]_[W2]
        RAMFILE_[W1]
        SWAP [W1]_[W2 RH] F_Q,B[W2 RH]_F
        [W2]-Q_F,F_Y,IX_MX               ;Test for overflow
        [BIT17]+[W1]_B,                  ;Previous byte number
         IF NOT CT [MZ] THEN [W1_RAMFILE]
        ZERO_[W1 LH]
        [K -1]+[W1]_B,                   ;Backup ramfile word
         GOTO [W1_RAMFILE]
```

```
.TOC    "       A (Address Break) Command"

;Here for a A "Address Break" command

.IF/FTADRB
A CMD:  CALL [GET CMD NUM]
        J[AB IF]_RAMFILE ADR
        [W1]_RAMFILE,
         CALL [GET CMD NUM]
        J[AB RD]_RAMFILE ADR
        [W1]_RAMFILE,
         CALL [GET CMD NUM]
        J[AB WR]_RAMFILE ADR
        [W1]_RAMFILE,
         GOTO [CMD PRSE8]
.ENDIF/FTADRB
```

```
.TOC    "       B (Bootstrap) Command"

;Here for a B "Bootstrap" command

B CMD:  IF [RUN] THEN [CMDERR NWR]
        GOTO [CMD PRSE8]
```

```
.TOC    "       C (Continue) Command"

;Here for a CONTINUE command

C CMD:  IF [RUN] THEN [CMDERR NWR]
        SET LOCAL,CALL [GET CMD PC]     ;Try to get a PC from command
        [BIT4]_[W1],                    ;(F.USR) Console now in user mode
         CALL [SET CMDFLG]
        CALL [SET PCI TXEN]             ;Start transmitter
        J[HALT CODE]_RAMFILE ADR
        ZERO_Y,ALU_RAMFILE,SET RUN,     ;We haven't halted
         CALL [SET PI MASK]             ; Enable interrupts
        GOTO [CMD PRSE8]

GET CMD PC:
        CALL [GET CMD NUM]              ;Get suggested starting address
        IF CT [UZ] RETURN
        [W1]_[PC],                      ;Set new PC
         RETURN
```

```
.TOC    "       D (Deposit) Command"

;Here for a D "Deposit memory" command

D CMD:  ZERO_[W1],                      ;No int recovery routine
         CALL [GET EXM ADR],O_UC        ;Get adr to deposit
DN:     CALL [UNMAPPED WRITE]
.IF/FTFCSL
        CALL [GET CMD NUM]              ;Get data
        [W1]_[MB],
         IF NOT CT [UZ] THEN [DN]
.ENDIF/FTFCSL
        GOTO [CMD PRSE8]

.IF/FTFCSL
JK CMD: CALL [GET JK NUM]               ;Get adr to deposit into
        [MB]_[PMA],
         IF CT [UZ] THEN [CMDERR NEA]   ;Require address
JK L:   CALL [GET JK NUM]               ;Get data to deposit
        IF CT [UZ] THEN [CMD PRSE8]
        J[EXM ADR]_RAMFILE ADR
        [PMA]_Y,ALU_RAMFILE,            ;Save last adr deposited
         CALL [UNMAPPED WRITE]
        [PMA]+1_[PMA],                  ;Next adr
         GOTO [JK L]

;Here to get an "Asciized" number
GET JK NUM:
        ZERO_[MB],1_UZ,
         CALL [GET CMD CHX]             ; Get 1st nonblank char from command
JK 1:   J[75]_[W2]
        [W1]-[W2]_Y,
         IF CT [IN] THEN [GCN 4]
        J[175]_[W2]
        [W1]-[W2]_Y,
         IF NOT CT [IN] THEN [GCN 4]
        [K 77]&[W1]_B,                  ;Strip extra bits
         J[2]_CTR
JK 2:   4*[MB]_B,LOOP [JK 2]
        [W1]+[MB]_B,O_UZ,
         CALL [GET CMD CHR]
        GOTO [JK 1]
.ENDIF/FTFCSL
```

```
.TOC    "       E (Examine) Command"

;Here for a E "Examine memory" command

E CMD:  J[E CMD RCOVR]_[W1]             ;Recovery routine for ints
        CALL [GET EXM ADR],1_UC         ;Get memory adr to examine
E CMD RCOVR:
        [PMA]_[PMA],MEM START READ,
         CALL [UNMAPPED READ]
EC 3:   [MB]_[W1],
         CALL [PUT CMR NUM]
        GOTO [CMD PRSE8]

;Here to get an address to examine or deposit
; Call  J[adr]_[W1]                     ;Interrupt recovery address
;       CALL [GET EXM ADR],0_UC         ;for deposit commands
;       CALL [GET EXM ADR],1_UC         ;For examine commands
;       Returns adr in [E] & [PMA], (and for deposits data in MB)
GET EXM ADR:
        [W1]_Y,                         ;Set recovery routine
         IF NOT CT [IZ] CALL [SET INT RCOVR]    ; for interrupts
        J[CMDERR E MER]_[W1]            ;Recovery routine for examines
        IF CT [UC] CALL [SET PF ME RCOVR]
        CALL [GET CMD NUM]              ;Get adr to examine
        J[EXM ADR]_RAMFILE ADR
        IF NOT CT [UZ] THEN [GEA2]      ;If no adr use last
        RAMFILE_[W1],                   ;Get last adr used
         IF NOT CT [UC] THEN [CMDERR NEA]       ;Require address
GEA2:   [W1]_[E],ALU_RAMFILE,           ;Save address in E and ramfile
         IF CT [UC] THEN [GEA6]         ; Branch for examine
        [E]_[PMA],                      ;Save adr in PMA also
         CALL [GET CMD NUM]             ;Get data
        [W1]_[MB],
         IF NOT CT [UZ] RETURN          ;Require data
        ;GOTO [CMDERR NEA]
CMDERR NEA:
        J[ERR.NEA]_[W1]                 ;Error code (not enough args)
        GOTOP [CMDERR]

GEA6:   [E]_[PMA],                      ;Save adr in PMA also
         CALL [PUT CMR CRLF]            ;Begin with a flourish
        [PMA]_[W1],                     ;Now type adr
         CALL [PUT CMR NUM]
        GOTO [PUT CMR SLSH]
```

```
.TOC    "       DI (Deposit I/O) Command"

;Here for an DI "Deposit I/O" command
.IF/FTFCSL
DI CMD: ZERO_[W1],                          ;No int recovery
         CALL [GET EXM ADR],O_UC            ;Get adr to deposit
        [W1]_[MEM-OP],                      ;Put data in right register
         CALL [WRIOX]                       ;Try to do it
        IF CT [UOVR] THEN [CMDERR NXA]
        GOTO [CMD PRSE8]
.ENDIF/FTFCSL
```

```
.TOC    "       EI (Examine I/O) Command"

;Here for an EI "Examine I/O" command
.IF/FTFCSL
EI CMD: ZERO_[W1],                          ;No int recovery
         CALL [GET EXM ADR],1_UC            ;Get adr to examine
        CALL [RDIOX]                        ;Do read like RDIO
        IF CT [UOVR] THENP [CMDERR NXA]
        [MEM-OP]_[W1],                      ;Copy data
         CALL [PUT CMR NUM]                 ; Display data
        GOTO [CMD PRSE8]
.ENDIF/FTFCSL
```

```
.TOC    "       DR (Deposit Ramfile) Command"

;Here for an DR "Deposit Ramfile" command
.IF/FTFCSL
DR CMD: ZERO_[W1],                      ;No int recovery
         CALL [GET EXM ADR],0_UC        ;Get adr to examine
        [PMA]_Y,Y_RAMFILE ADR
DRC 3:  [MB]_RAMFILE,                   ;Deposit ramfile
         GOTO [CMD PRSE8]

;.TOC   "       ER (Examine Ramfile) Command"

;Here for an ER command
ER CMD: ZERO_[W1],                      ;No int recovery
         CALL [GET EXM ADR],1_UC        ;Get adr to examine
        [PMA]_Y,Y_RAMFILE ADR           ;Address ramfile
        RAMFILE_[W1],NRFRD/1,
         CALL [PUT CMR NUM]
        GOTO [CMD PRSE8]
.ENDIF/FTFCSL
```

```
.TOC    "       DV (Deposit Virtual) Command"

;Here for an DV "Deposit Virtual" command
.IF/FTFCSL
DV CMD: J[DV CMD RCOVR]_[W1],POP        ;Recovery routine for ints
        CALL [GET EXM ADR],0_UC         ;Get adr to examine
DV CMD RCOVR:
        SET GLOBAL,CALL [MEMORY WRITE]  ;Write virtual
        GOTO [CMD PRSE8]

;.TOC   "       EV (Examine Virtual) Command"

;Here for an EV command
EV CMD: J[EV CMD RCOVR]_[W1]            ;Recovery routine for ints
        CALL [GET EXM ADR],1_UC         ;Get adr to examine
EV CMD RCOVR:
        SET GLOBAL,CALL [MEM READ 0]    ;Read virtual
        GOTO [EC 3]                     ;Rest like E cmd
.ENDIF/FTFCSL
```

```
.TOC    "       H (Halt) Command"

;Here for a HALT command

H CMD:  ;IF [NOT RUN] THEN [CMDERR NWR] ;This is silly if not running
        J[HALT CONSOLE]_[W1]            ;Stopped because told to
SET HALT CODE:
        J[HALT CODE]_RAMFILE ADR
        [K 77]_Y,Y_2914 MASK,           ;Stop interrupts
         IF [NOT RUN] THENP [WRITE HSB]
        [W1]_RAMFILE,CLEAR RUN,         ;Save code and go type result
         GOTOP [WRITE HSB]
```

```
.TOC    "       I (Initialize) Command"

;Here for a INITIALIZE command

I CMD:  IF [RUN] THEN [CMDERR NWR]
        CALL [SYS RESET]
        GOTO [CMD PRSE8]

SYS RESET:
        ZERO_[PC FLAGS],CLEAR USER      ;Initialize register
        ZERO_[EPT],CLEAR PAGED          ;Initialize register
        ZERO_[UPT],SET LOCAL            ;Initialize register
        ZERO_[SPT],CLEAR PXCT           ;Initialize register
;Set up PROC REG
        J[33B MASK]_RAMFILE ADR         ;Want high order 3 bits on
.IF/FT1OPAG
        RAMFILE_[W1],NRFRD/1,CLEAR TOPS20
.ENDIF/FT1OPAG
.IFNOT/FT1OPAG
        RAMFILE_[W1],NRFRD/1,SET TOPS20
.IFNOT/FT2OPAG
        FT1OPAG=0 & FT2OPAG=0           ;This won't work
.ENDIF/FT2OPAG
.ENDIF/FT1OPAG
        FJ[0200]_[W1],                  ;Line PI level = 1
         COMPLEMENT [W1]_F              ; 700000,,200
        .![PROC REG]_RAMFILE ADR        ;Address ramfile copy of proc reg
        [W1]_RAMFILE,CLEAR TRAP,
         CALL [RESET PI]
        GOTO [RESET IO]                 ;Perform an I/O reset
```

```
.TOC    "       R (Repeat) Command"

;Here for a REPEAT command

REPEAT: CALL [GET CMD NUM]              ;Get repeat count
        J[CMDRPT]_RAMFILE ADR           ;Address repeat counter
        NEGATE [W1]_F,F_Q_Y,IX_MX,
         CALL [RAMFILE_W1]              ;Get current value for counter
        [W1]+1_Y,ALU_RAMFILE,           ;Update repeat counter
         IF CT [IZ IX_MX] THEN [CMD PRSE8]
        [W1]_Y,
         IF CT [IN] THEN [REPEAT 3]
        Q_Y,ALU_RAMFILE                 ;Reseed repeat counter
REPEAT 3:
        CALL [ENABLE PARSE]
        GOTO [CMD PRSE8]

RAMFILE_W1:
        RAMFILE_[W1],NRFRD/1,
         RETURN
```

```
.TOC    "       S (Start) Command"

;Here for a START command

S CMD:  IF [RUN] THEN [CMDERR NWR]
        CALL [SYS RESET]                ;Reset lots of things
        GOTO [C CMD]                    ;Rest is like a C command
```

```
.TOC    "       SH (Shut Down) Command"

SH CMD: J[30]_[PMA]
        ONES_[MB],                          ;Write -1 in location 30
         CALL [UNMAPPED WRITE]
        GOTO [CMD PRSE8]
```

```
.TOC    "       SI (Single Instruction) Command"

SI CMD: IF [RUN] THEN [CMDERR NWR]
        SET LOCAL,CALL [GET CMD PC]     ;Try to pick up a PC
        ZERO_[W1],                      ;Memory errs & page fails
         CALL [SET PF ME RCOVR]         ; are usual
        J[HALT CODE]_RAMFILE ADR        ;Address halt code
        J[HALT SI]_[W2]                 ;Single instruction halt
        J[SI I RCOVR]_[W1]
        [W2]_RAMFILE,                   ;Set Halt code
         CALL [SET INT RCOVR]
        CALL [SET PI MASK]              ;Allow interrupts
        ZERO_[W1],                      ;Normal recovery for
         CALL [SET PF ME RCOVR]         ; for mem errs & page fails
SI RCOVR:
        [PC]_Q_[E],                     ;Put PC in ram address & Q
         SPEC SEL/PAGE TABLE ENTRY,
         GOTOP [IFETCH 0]

;Here if got interrupted out of SI cmd
SI I RCOVR:
        [K -1]+[PC RH]_B,               ;Backup PC
         GOTO [SI RCOVR]

;Here to set exit for interrupt service
SET INT RCOVR:
        J[INT RCOVR]_RAMFILE ADR        ;Address ramfile
        [W1]_RAMFILE,RETURN
```

```
.TOC    "       SM (Start u-code) Command"

;Here for a START u-CODE command
.IF/FTSM
SM CMD: CALL [GET CMD NUM]
        [W1]_Y,GOTO Y                   ;Go whither he asketh
.ENDIF/FTSM
```

```
.TOC    "       T (Test) Command"

;Here for a SELF TEST command

T CMD:  IF [RUN] THEN [CMDERR NWR]
        GOTO [TEST]
```

```
;Here to write [W1] into Ramfile location Q+1
RF WRITE:
        Q+1_F,F_Q_Y,Y_RAMFILE ADR,      ;Point Ramfile at right location
         GOTO [W1_RAMFILE]              ;Write Ramfile

.IF/FTDIAG
;Here to load LEGAL SECTION, SECTION 0, AC REF, CONTEXT MATCH in W1
; Call with expected page table data in W2, expected latches in W3
RDPG LATCHES:
        CALL [TVR XX]                   ;Check data expected is obtained
        ZERO_[W1],
         IF [LEGAL SECTION] CALL [W1+1_W1]
        2*[W1]_[W1],
         IF [SECTION 0] CALL [W1+1_W1]
        2*[W1]_[W1],
         IF [AC REF] CALL [W1+1_W1]
        2*[W1]_[W1],
         IF [CONTEXT MATCH] CALL [W1+1_W1]
        [W1].XOR.[W3]_Y,
         IF CT [IZ] RETURN
        [W3]_[W2],
         GOTO [ALU BROKE]

SET TOPS20 LATCH:
        SET TOPS20,RETURN
SET TRAP LATCH:
        SET TRAP,RETURN
SET PXCT LATCH:
        SET PXCT,RETURN
SET RUN LATCH:
        SET RUN,RETURN
SET LOCAL LATCH:
        SET LOCAL,RETURN

W1+1_W1:
        [W1]+1_[W1],                    ;Flag is set
         RETURN

;Here to initialize Q with first ramfile adr
TVR XO: J[1777]_[W1],Y_RAMFILE ADR      ;Highest adr in Ramfile
        [W1]_Q,RETURN

;Here to verify ramfile contents are correct
TVR XX: RAMFILE_[W1],NXFRD/1,
         IX_MX,LCTXT/1
TVR XY: [W1].XOR.[W2]_Y,
         IF CT [IZ] RETURN
        GOTO [RF BROKE]
.ENDIF/FTDIAG
```

```
.TOC    "       ZM (Zero memory) Command"

;Here for a Zero Memory command
.IF/FTZM
ZM CMD: J[ZMC 5]_[W1]                   ;Routine for memory errors
        ZERO_[MB],
         CALL [SET PF ME RCOVR]
        [K -1]_[PMA]
ZM 10:  J[EXM ADR]_RAMFILE ADR
        [PMA]+1_[PMA],ALU_RAMFILE,
         CALL [UNMAPPED WRITE]
        [PMA]_[PMA],MEM START READ,
         CALL [UNMAPPED READ]
        [MB]_Y,IF CT [IZ] THEN [ZM 10]
        GOTO [CMDERR MER]

ZMC 5:  [K -1.0]&[W1]_Q                 ;Get only code
        Q.XOR.[BIT17]_F,F_Y,
         IF CT [IZ] THEN [CMD PRSE8]
        GOTO [CMDERR MER]
.ENDIF/FTZM
```

```
;Here on incorrect u-code dispatch
MCE:    ;GOTO [U-CODE ERR]

;Here on a horrible u-code problem
U-CODE ERR:
        J[HALT DSP]_[W1]                ;Illegal u-code dispatch
        GOTOP [SET HALT CODE]

;Here if the ALU is broken
ALU BROKE:
;Here if the Ramfile (or addressing logic) is broken
; Bad data in W1, good data in W2, test # on Y bus
RF BROKE:
        [PXCT]_Q                        ;Test code

; W1 on A bus, W2 on B bus, test # on Y bus
BROKE9: R SEL/W1,B SEL/W2,O_CO,Q+C_F,F_Y,
         GOTO [BROKE9]
```

```
.TOC    "       WRITE HALT STATUS BLOCK"

;Here when ISP code halts
WRITE HSB:
        [K 77]_Y,Y_2914 MASK,POP        ;Disallow interrupts
        [K 7777.-1]&[PC]_B,POP          ;Mask PC to 30 bits
        J[HSB PMA]_RAMFILE ADR,POP      ;Save PMA here
        [PMA]_RAMFILE,POP               ;Save PMA
        J[HSB]_RAMFILE ADR              ;Get adr of Halt Status Block
.IF/FTCRAM
        RAMFILE_[PMA],NRFRD/1
        [PMA]_[PMA],                    ;Let latches go
         IF CT [IZ] THEN [WHSB2]        ;No block = don't write
.ENDIF/FTCRAM
.IFNOT/FTCRAM
        RAMFILE_[PMA],NRFRD/1,
         IF CT [IZ] THEN [WHSB2]        ;No block = don't write
.ENDIF/FTCRAM
        J[HSB MB]_RAMFILE ADR           ;Save MB here
        J[HSB MER]_[W1]                 ;Recovery in case of memory errs
        [MB]_RAMFILE,
         CALL [SET MER]
        ZERO_[W1],IF [LOCAL] CALL [W1+1_W1]
        2*[W1]_[W1],IF [USER] CALL [W1+1_W1]
        2*[W1]_[W1],IF [PAGED] CALL [W1+1_W1]
        2*[W1]_[W1],IF [RUN] CALL [W1+1_W1]
        2*[W1]_[W1],IF [TOPS20] CALL [W1+1_W1]
        2*[W1]_[W1],IF [PXCT] CALL [W1+1_W1]
        2*[W1]_[W1],IF [TRAP] CALL [W1+1_W1]
        [W1]_[MB],CALL [UNMAPPED WRITE]
        [W2]_[MB],CALL [UNMAPPED WRITE NEXT]
        [W3]_[MB],CALL [UNMAPPED WRITE NEXT]
        [W4]_[MB],CALL [UNMAPPED WRITE NEXT]
        [W5]_[MB],CALL [UNMAPPED WRITE NEXT]
        [W6]_[MB],CALL [UNMAPPED WRITE NEXT]
        [IR]_[MB],CALL [UNMAPPED WRITE NEXT]
        [PC]_[MB],CALL [UNMAPPED WRITE NEXT]
        [E]_[MB],CALL [UNMAPPED WRITE NEXT]
        J[HSB MB]_RAMFILE ADR           ;Get saved MB
        RAMFILE_[MB],NRFRD/1,CALL [UNMAPPED WRITE NEXT]
        [AC-OP]_[MB],CALL [UNMAPPED WRITE NEXT]
        [MEM-OP]_[MB],CALL [UNMAPPED WRITE NEXT]
        [PXCT]_[MB],CALL [UNMAPPED WRITE NEXT]
        [K 77]_[MB],CALL [UNMAPPED WRITE NEXT]
        [K 777]_[MB],CALL [UNMAPPED WRITE NEXT]
        [K -1.0]_[MB],CALL [UNMAPPED WRITE NEXT]
        [PC FLAGS]_[MB],CALL [UNMAPPED WRITE NEXT]
        [AC-OP+1]_[MB],CALL [UNMAPPED WRITE NEXT]
        [MEM-OP+1]_[MB],CALL [UNMAPPED WRITE NEXT]
        [EPT]_[MB],CALL [UNMAPPED WRITE NEXT]
        [UPT]_[MB],CALL [UNMAPPED WRITE NEXT]
        [SPT]_[MB],CALL [UNMAPPED WRITE NEXT]
        J[HSB PMA]_RAMFILE ADR          ;Get saved PMA
        RAMFILE_[MB],NRFRD/1,CALL [UNMAPPED WRITE NEXT]
        [TIME]_[MB],CALL [UNMAPPED WRITE NEXT]
        [PI REG]_[MB],CALL [UNMAPPED WRITE NEXT]
        [BIT4]_[MB],CALL [UNMAPPED WRITE NEXT]
        [BIT12]_[MB],CALL [UNMAPPED WRITE NEXT]
        [BIT17]_[MB],CALL [UNMAPPED WRITE NEXT]
```

```
        [CO!C1]_[MB],CALL [UNMAPPED WRITE NEXT]
        [K 7777.-1]_[MB],CALL [UNMAPPED WRITE NEXT]
        [K 407777.0]_[MB],CALL [UNMAPPED WRITE NEXT]
        [K -1]_[MB],CALL [UNMAPPED WRITE NEXT]
        J[HALT CODE]_[W5],              ;Address halt code in ramfile
         Y_RAMFILE ADR
        J[11]_[W6]                      ;Also want 9 more locations
        RAMFILE_[MB],
         CALL [UNMAPPED WRITE NEXT]
WHSB1:  [W5]+1_[W5],Y_RAMFILE ADR       ;Next ramfile location
        RAMFILE_[MB],NRFRD/1
        ZERO_Y,ALU_RAMFILE,
         CALL [UNMAPPED WRITE NEXT]
        [K -1]+[W6]_B,
         IF NOT CT [IZ] THEN [WHSB1]

;Type <CR><LF>?H<halt code> <flags> <PC>
WHSB2:  ZERO_[W1],CALL [SET INT RCOVR]  ;In case was SI cmd
        [BIT4]J[F.O!F.S]_[W1],          ;(F.USR) Flags to clear
         SET LOCAL
        CALL [CLEAR CMDFLG]
        J[110]_[W6]                     ;Ascii "H"
        CALL [PUT ? W6]
        J[HALT CODE]_RAMFILE ADR
        RAMFILE_[W1],NRFRD/1,           ;Get halt code from ramfile
         CALL [PUT CMR NUM]             ; and type it
        CALL [PUT CMR SPACE]            ;Type a space
        [PC FLAGS]_[W1],                ;Type flags
         CALL [P6DNUM]
        CALL [PUT CMR SPACE]
        [PC]_[W1],
         CALL [PUT CMR NUM]
        GOTO [CONSOLE PROMPT]

;Here in case of memory errors while writing HSB
HSB MER:
        J[HALT HSBMER]_[W1]
        J[HALT CODE]_RAMFILE ADR
        [W1]_RAMFILE,
         GOTO [WHSB2]

W1+1_W1:
        [W1]+1_[W1],                    ;Flag is set
         RETURN
```

```
.TOC    "Instruction fetch"

IFETCH:
.IF/DEBUG2
        IF [GLOBAL] THEN [U-CODE ERR]
.ENDIF/DEBUG2
.IF/SH&TL
        CALL [SHOW&TELL]
.ENDIF/SH&TL
        [PC]_Q_[E],                     ;Put PC in ram address & Q
         SPEC SEL/PAGE TABLE ENTRY,
         IF [NOT RUN] THENP [WRITE HSB]
IFETCH 0:
        Q&[K 777]_Q RAMFILE_[W1],NRFRD/1,       ;Put on page adr in Q
         CLEAR PXCT,IX_MX,LCTXTM/1,     ; Put page entry in W1
         IF [UNPAGED OR AC] THENP [IFETCH 1]
        Q.OR.[W1]_[PMA],                ;Start memory
         MEM START READ,
         IF [CONTEXT MATCH] THENP [IFETCH 3]
IFETCH 05:
        [PC]+1_[PC RH],                 ;Increment PC
         CALL [MEM READ 0]
        GOTOP [IFETCH 4]

;Here if not paged or AC REF
IFETCH 1:
        [E]_[PMA],SPEC SEL/VMA,         ;Address AC in Ramfile
         IF [AC REF] THENP [IFETCH 05]
        [E]_[PMA],MEM START READ        ;Start memory

;Here if not paged or paging done
IFETCH 3:
        [PC]+1_[PC RH],MEM HOLD,        ;Increment PC
         CHECK INTERRUPTS
        MEM_IR_[IR],SPEC SEL/XR         ;Read memory into IR
                                        ; Set XR=0 and @ flops
        [IR]_[E RH],MEM HOLD,
         IF [NO BUS ERROR] THENP [IFETCH EFA]
        [K -1]+[PC RH]_B,               ;Backup the PC
         GOTO [IFETCH 05]               ; And try again

;Here on an XCT, PXCT, LUUO
IFETCH 4:
        [MB]_[IR],SPEC SEL/XR,          ;Set XR=0 and @ flops
         LOAD IR

;Here with SPEC SEL/XR to set XR=0 and @ flags
IFETCH EFA:
        [IR]_[E RH],SEL AC+[0],         ;Y rh_E rh
         IF [NOT (I OR XR)] D [OPERAND FETCH]   ;Dispatch on op code to get operands
        [IR]_Y,SPEC SEL/XR,             ;Do effective adr calc
         CALL [EFA 2]
        SEL AC+[0],
         DISPATCH [OPERAND FETCH]       ;Dispatch on op code to get operands

;Here if instruction being done under PXCT
IFETCH PXCT:
        J[0400]_[PXCT RH]               ;PXCT bit for effective adr calc
        [K 7777.-1]&[E]_B,SET LOCAL,
```

```
 CALL [SET PXCT CTXT]            ;Set context
[IR]_[E RH],SPEC SEL/XR          ;Y rh_E rh
IF [I OR XR] CALL [EFA 2]
CALL [SET CURRENT CTXT]
SEL AC+[O],                      ;Dispatch on op code to get operands
 DISPATCH [OPERAND FETCH]
```

```
.TOC    "EFFECTIVE ADDRESS CALCULATION"

;Here to do an effective address calculation
; Call: x_[MB],SPEC SEL/XR
; Returns       adr in E
;               clobbers W1
EFA CALC:
        [MB]_[E RH],SET LOCAL,          ;Y rh_E rh
         IF [NOT (I OR XR)] RETURN

EFA 2:  [K 407777.0]_Q RAMFILE_[W1],NRFRD/1,    ;Get index register
         IF [NOT INDEXED] THEN [FETCH IW]
        RAMFILE_[MB]
        Q&[W1]_Y,IX_MX,                 ;Check section # in IDX
         IF [SECTION 0] THEN [EFA LCL INDEX]
        [K -1.0]BAR&[E]_Q,              ;Put E rh in Q
         IF CT [(MNxMOVR)!MZ] THEN [EFA LCL INDEX]
        Q+[W1]_[E],SET GLOBAL,          ;IDX(6-35)+Y(18-35)_E(6-35)
         IF [NOT B BIT 18] THEN [EFA 4]
        [K -1.0]+[E]_B,                 ;Sign extend Y
         IF [NOT INDIRECT] RETURN
        GOTO [FETCH IW]
EFA 4:  IF [NOT INDIRECT] RETURN
        GOTO [FETCH IW]

EFA LCL INDEX:
        [W1]+[E RH]_B,                  ;Y rh+IDX rh_E rh
         SET LOCAL,
         IF [NOT INDIRECT] RETURN
        ;GOTO [FETCH IW]

FETCH IW:
        READ [E]                        ;Fetch indirect word
                                        ; and SPEC SEL/IW
.IFNOT/FT2OPAG
        GOTO [EFA CALC]
.ENDIF/FT2OPAG
.IF/FT2OPAG
        RAMFILE_[W1],NRFRD/1,           ;Get index register
         IF [SECTION 0] THEN [EFA CALC]
        CLEAR LOCAL,
         IF [ILLEGAL IW] THENP [PF X24]
        IF [LOCAL IW] THEN [EFA CALC]
        [MB]_[E],
         SPEC SEL/PAGE TABLE ENTRY,
         IF [NOT (I OR XR)] RETURN
        [MB]+[W1]_Q,                    ;XR(6-35)+IW(6-35)_Q
         IF [NOT INDEXED] THEN [FETCH IW]
        Q_[E],
         SPEC SEL/PAGE TABLE ENTRY,
         IF [NOT INDIRECT] RETURN
        GOTO [FETCH IW]
.ENDIF/FT2OPAG
```

```
.TOC    "OPERAND FETCH"

;These routines fetch instruction operands
;       if single operand is put in MEM-OP
;       AC & mem operands are put in AC-OP and MEM-OP
;       IX_MX for MEM-OP
;       IX_UX for AC-OP

;Double AC fetch - AC_MEM-OP, AC+1_MEM-OP+1
DFETCH AC:
        RAMFILE_[MEM-OP],NRFRD/1,       ;Fetch first AC
         IX_MX,                         ; Latch IZ in MZ
         SEL AC+[1]                     ; Point RAMFILE at AC+1
        RAMFILE_[MEM-OP+1],NRFRD/1,     ;Get 2nd AC from RAMFILE
         SEL AC+[0],                    ; Repoint RAMFILE at AC
         DISPATCH [INST EXCT]

;Double AC & Mem fetch
DFETCH AC&MEM:
        RAMFILE_[AC-OP],NRFRD/1,        ;Get 1st AC
         SEL AC+[1]
        RAMFILE_[AC-OP+1],NRFRD/1,      ;Get 2nd AC
         GOTO [DFETCH MEM]

;Double word fetch from memory
; Sets MX according to highorder operand
DFETCH MEM:
        [K 7777.-1]&[E]_F,F_Q_B,        ;Get 1st operand
         SPEC SEL/PAGE TABLE ENTRY,
         IF [PXCT] THEN [PXCT DFETCH]
        CALL [MEMORY READ]
        [MB]_[MEM-OP],                  ;Save 1st operand
         CALL [READ NEXT]
        [MB]_[MEM-OP+1],                ;We don't need to backup E
         GOTO [DFM 8]                   ; because noone uses it again
PXCT DFETCH:
        J[0200]_[PXCT RH]               ;Bit for this flavour PXCT
        CALL [SET PXCT CTXT]
        [K 7777.-1]&[E]_F,F_Q_B,        ;Get 1st operand
         SPEC SEL/PAGE TABLE ENTRY,
         CALL [MEMORY READ]
        [MB]_[MEM-OP],                  ;Save 1st operand
         CALL [READ NEXT]
        [MB]_[MEM-OP+1],
         CALL [SET CURRENT CTXT]
DFM 8:  [MEM-OP]_Y,IX_MX,SEL AC+[0],    ;Set flags
         DISPATCH [INST EXCT]
```

```
;Single AC and memory word fetch
FETCH AC&MEM:
        RAMFILE_[AC-OP],NRFRD/1,        ;Get AC from RAMFILE
         GOTO [FETCH MEM]

;Single word fetch from memory and set MX according to data
FETCH MEM:
        [E]_Q_[PMA],                    ;Read memory operand if not PXCT
         SPEC SEL/PAGE TABLE ENTRY,
         IF [NOT PXCT] CALL [MEMORY READ]
        [MB]_F,F_Q_[MEM-OP],            ;Copy to right reg and set flags
         IX_MX,SEL AC+[0],
         IF [NOT PXCT] D [INST EXCT]
        CALL [PXCT FETCH 200]           ;Get data
        [MB]_F,F_Q_[MEM-OP],            ;Copy to right reg and set flags
         IX_MX,SEL AC+[0],
         DISPATCH [INST EXCT]

;Single AC and memory word fetch
FETCH AC&MEM(W):
        RAMFILE_[AC-OP],NRFRD/1,        ;Get AC from RAMFILE
         GOTO [FETCH MEM(W)]

;Single word fetch and verify writable from memory
FETCH MEM(W):
        [E]_Q_[E],                      ;Read memory operand if not PXCT
         SPEC SEL/PAGE TABLE ENTRY,
         IF [NOT PXCT] CALL [MEMORY READ]
        [MB]_F,F_Q_[MEM-OP],            ;Copy to right reg and set flags
         IX_MX,
         IF [NOT PXCT] THEN [FM(W) 2]
        CALL [PXCT FETCH 200]           ;Fetch data
        [MB]_F,F_Q_[MEM-OP],            ;Copy to right reg and set flags
         IX_MX,SEL AC+[0]
FM(W) 2:
        B SEL/W1,SEL AC+[0],            ;Check writable from paging ram
         IF [B BIT 3] D [INST EXCT]
        IF [UNPAGED OR AC] D [INST EXCT]
        CALL [M W XX]                   ;Try to make writable
        SEL AC+[0],DISPATCH [INST EXCT]
```

```
;Single AC fetch and immediate operand
FETCH AC&I:
        RAMFILE_[AC-OP],NRFRD/1,        ;Get AC from ramfile
         GOTO [FETCH I]

;Immediate mode operation
FETCH I:
        SWAP [K -1.0]_[MEM-OP]          ;Mask for immediate operand
        [E]&[MEM-OP]_F,F_Q_B,IX_MX,     ;Get immediate operand
         SEL AC+[0],
         DISPATCH [INST EXCT]

;Single AC fetch but don't set Mx flags
FETCH AC:
.IF/FAST
        RAMFILE_[MEM-OP],NRFRD/1,       ;Get AC from RAMFILE
         DISPATCH [INST EXCT]
.ENDIF/FAST

;Single AC fetch and set Mx flags
FETCH AC(MX):
        RAMFILE_[MEM-OP],NRFRD/1        ;Get AC from RAMFILE
        [MEM-OP]_[MEM-OP],IX_MX,        ;Set Mx flags
         DISPATCH [INST EXCT]
```

```
;Fetch AC and fetch IO word
FETCH AC&IO:
        RAMFILE_[AC-OP],NRFRD/1          ;Get AC from RAMFILE

;Fetch IO word
FETCH IO:
        CALL [RDIOX]                     ;Get data
        SEL AC+[0],                      ; Repoint RAMFILE at AC
         IF NOT CT [UOVR] D [DISPATCH 3]
        GOTO [IO PF]

;Here to do an IO write
; returns with UOVR if illegal
RDIOX:  [K 7777.-1]&[E]_B,1_UOVR,        ;Strip extraneous bits
         CALL [VAL IO ADR]               ; Validate IO address
        [E]_[PMA],START IO READ,         ;Send IO address
         IF CT [MN] THEN [RDIO TTY]
        IO_[MEM-OP]                      ;Perform transfer
        0_UOVR,IF [NO BUS ERROR] RETURN
.IFNOT/FTCKBP
        IF [NOT MEM EXISTS] THEN [SET UOVR]
        IF [NOT MEM FAULT] RETURN
.ENDIF/FTCKBP
;Here to set the u-overflow flag
SET UOVR:
        1_UOVR,RETURN                    ;Set flag and exit

;Here to validate IO address
; Call with address in E
;       If tty returns IN, address in W6, TTYRCV in W1, 7 in W2
VAL IO ADR:
.IF/FTMBZ
        J[7760]_[W1]                     ;Build mask for address
        SWAP [W1]_[W1],1_UOVR            ;We are checking bits 6-13 of E
        [W1]&[E]_Y,                      ;If nonzero illegal
         IF NOT CT [IZ] THENP [VIA9]     ;If illegal POP, POPJ
.ENDIF/FTMBZ
        J[LNZSW]_[W1]                    ;First non tty adr
VIA9:   [E]-[W1]_Y,
         IF NOT CT [IN IX_MX] RETURN
        [E]_[W6],Y_RAMFILE ADR           ;Address ramfile
        LRS [W6]_[W3],PUSH J[1]_CTR      ;Copy & shift address
        LRS [W3]_[W3],RFCT               ;Shift right
        J[7]_[W2]
        [W2]BAR&[W6]_B                   ;Make adr of status word
        [W2]&[W3]_B                      ;Leave only line number in W3
        SWAP [W3]_[W6 LH]
        J[TTYRCV BITS]_[W1],             ;Min legal address
         RETURN

;Here when reading TTY register
RDIO TTY:
        [E]-[W1]_F,F_Q_Y,
         IF CT [IN IX_MX] RETURN
        RAMFILE_[MEM-OP],                ;Get data from ramfile
         IF CT [MZ] THEN [RDIOTM]
        0_UOVR
        [W2]&[W6]_F,F_Q_Y,               ;Leave only word offset
```

```
        IF NOT CT [IZ] RETURN           ;If not reading status done

;Here if reading the status register
RDIOTO: CALL [RD PCI SR]
        J[300]_[W2]                     ;Mask for SR7&SR6
        [W2]&[W1]_B,                    ;Leave only SR7&SR6
         CALL [SWAP W2_W2]
        [W2]BAR&[MEM-OP]_B,             ;Clear old SR7&SR6
         CALL [SWAP W1_W1]
        [W1].OR.[MEM-OP]_B,
         RETURN

.IF/FTTTYF
        RAMFILE_[MEM-OP],O_UOVR,        ;Get received char
         CALL [ZERO_RAMFILE]
        [W1]BAR&[W6]_B,Y_RAMFILE ADR    ;Make adr of line block
        RAMFILE_[W5],NRFRD/1,           ;Get current status
         CALL [CLR RDN]
        GOTO [SET PROC REG]             ;Be sure int flags are correct
.ENDIF/FTTTYF

;Here to read TTYRCV BITS
RDIOTM: RAMFILE_[MEM-OP],O_UOVR,
         RETURN
```

```
.TOC    "INSTRUCTION EXECUTION"

;Here for LUUO's have already been through CHK PC SECT
LUUO:   IF [SECTION 0] THEN [LUUO0]     ;Check PC section
        IF [EXEC] THEN [MUUO]
        J[420]_[PMA]                    ;Offset into UPT for LUUO block
        [UPT]+[PMA]_B,MEM START READ,   ;Location which has block adr
         CALL [UNMAPPED READ]
        [MB]_[PMA],SET GLOBAL,          ;Save adr in PMA
         CALL [SV UUO STS]
        WRITE [PMA]                     ;Write flags,code,ac
        [PC]_[MB],CALL [WRITE NEXT]     ;Write 30 bit PC
        [MEM-OP]_[MB],CALL [WRITE NEXT] ;Write 30 bit E
        [E]+1_[E],                      ;Fetch new PC
         SPEC SEL/PAGE TABLE ENTRY,
         CALL [MEMORY READ]
        [MB]_[PC],SET LOCAL,
         GOTOP [IFETCH]

;Here if LUUO is done in section 0
LUUO0:  SWAP [IR]_[MB],                 ;Put opcode A in MB rh
         CALL [LUUO 2]
        SWAP [E]_[MB LH],               ;Want E in RH
         CALL [SWAP MB_MB]
        J[40]_[E]
        [E]_[PMA],
         IF [EXEC] THEN [LUUO0E]
        WRITE [E]                       ;Save info
        [E]+1_0_[E],                    ;Get instruction to execute
         SPEC SEL/PAGE TABLE ENTRY,
         CALL [MEMORY READ]
        GOTOP [IFETCH 4]
LUUO0E: [EPT]+[PMA]_B,                  ;Save opcode and E
         CALL [UNMAPPED WRITE]
        [PMA]+1_[PMA],MEM START READ,
         CALL [UNMAPPED READ]
        GOTO [IFETCH 4]

SWAP MB MB:
        SWAP [MB]_[MB],                 ;Put Opcode in LH
         RETURN
```

```
;Here as part of MUUO or LUUO to put [flags,,opcd],[PC]_F,[E] in MB
; Strips PC to 30 bits, puts E in MEM-OP
SV UUO STS:
        SWAP [IR]_[MB],                 ;Put opcode A in MB rh
         CALL [LUUO 2]                  ;Put 0,,opcode A in MB
        [PC FLAGS]_[MB LH]
        [K 7777.-1]&[PC]_B              ;Strip PC to 30 bits
E_MEM-OP:
        [E]_[MEM-OP],SECTION SELECT     ;Preserve RH of AC
        [K 7777.-1]&[MEM-OP]_B,         ;Strip extraneous bits
         IF [NOT AC REF] RETURN
        [PC]_Y,SECTION SELECT,          ;Check PC section
         IF [GLOBAL] RETURN
        IF [SECTION 0] RETURN
        [BIT17]_[MEM-OP LH],            ;Sect = 1 here
         RETURN

;Here to put 0,,opcode A in MB
LUUO 2: J[37]_[W1]                      ;Get mask for OPcode and AC field
        [W1]BAR&[MB]_B,                 ;Clear extraneous bits
         RETURN
```

```
.NOBIN
;                       TOPS20                              TOPS10
;           +-----------------------------------+    +-----------------------------------+
;       424 ! flags       ! 0 ! opcode ! A ! OO !    ! opcode ! AC ! OO !       E        !
;           +-----------------------------------+    +-----------------------------------+
;       425 ! OO !      PC                        !    !  flags           !       PC       !
;           +-----------------------------------+    +-----------------------------------+
;       426 ! OO !      E                         !    !  process context word             !
;           +-----------------------------------+    +-----------------------------------+
;       427 ! process context word                !
;           +-----------------------------------+
.BIN

;Here for MUUO's
; Will do several 2910 pops just in case
MUUO:   J[424]_[PMA],POP                ;Offset into UPT for block
        [UPT]+[PMA]_B,                  ;Physical adr of block
         CALL [SV UUO STS]
.IF/FT2OPAG
.IF/FT1OPAG
        IF [TOPS20] THEN [MUUO 1]
.ENDIF/FT1OPAG
.ENDIF/FT2OPAG
.IF/FT1OPAG
        SWAP [MB]_[MB]                  ;Other side please
        [E]_[MB RH],
         CALL [UNMAPPED WRITE]
        [PC FLAGS]_[MB]
        [PC]_[MB RH],
         CALL [UNMAPPED WRITE NEXT]
        CALL [GET CONTEXT WD]
        CALL [UNMAPPED WRITE NEXT]
        [PMA]+1_[PMA],                  ;Point to 427
         GOTO [MUUO 2]
.ENDIF/FT1OPAG

;Here for TOPS20 MUUO
.IF/FT2OPAG
MUUO 1: LK 7777,,-1]&[PC]_B,            ;Strip PC to 30 bits
         CALL [UNMAPPED WRITE]          ;Write flags,,op cd,ac
        [PC]_[MB],                      ;Write 30 bit PC
         CALL [UNMAPPED WRITE NEXT]
        [MEM-OP]_[MB],                  ;Write 30 bit E
         CALL [UNMAPPED WRITE NEXT]
        CALL [GET CONTEXT WD]           ;Get context word next
        [PMA]_Q_[PMA],                  ;Write context word in block
         CALL [UNMAPPED WRITE NEXT]
.ENDIF/FT2OPAG
;Here to get new PC on an MUUO
MUUO 2: ZERO_[PC FLAGS],                ;Clear flags
         IF [EXEC] THEN [MUUO 4]
        J[4]_[W1],POP,SET EXEC
        [W1]+[PMA]_B,POP
        J[4000]_[W1]                    ;Bit for previous context user
        SWAP [W1]_[PC FLAGS]
MUUO 4: [PMA]+1_[PMA],MEM START READ,
         CALL [UNMAPPED READ]           ; Get new PC
        [MB]_[PC].SET LOCAL,
```

```
 IF [TOPS20] THEN [IFETCH]
[?C FLAGS].OR.[MB]_B,           ;Add prev context user
 GOTO [MB_PC FLAGS]
```

```
;Here for DMOVNx
DMOVNX: 2*[MEM-OP+1]_Q                  ;Put low order bits in Q
        NEGATE Q_F,F_Q_Y,IX_MX          ;Negate low order bits
        [MEM-OP]BAR+C_F,MC_C IX_UX,     ;Negate high order bits
         F_[MEM-OP]
        Q_F,LRS F_B,B SEL/MEM-OP+1,     ;Save loworder portion
         O_SION O_QION,
         IF NOT CT [MZ] D [OPERAND STORE]
        IF CT [UZ] THEN [SET CO!C1 STORE]
        IF NOT CT [UOVR] D [OPERAND STORE]
        GOTO [SET OV!C1!T1 STORE]
```

```
;Here after P&S SETUP for ADJBP
ADJBP0: [AC-OP]_[AC-OP+1],IX_MX,          ;Copy # of bytes to adjust
         IF [AC.EQ.0] THEN [TO NOWHERE] ;In case was really IBP
        [W6]_Y,                          ;Check for S=0
         IF CT [MN IX_MX] THEN [ADJ04]
        ZERO_[AC-OP],GOTO [ADJ06]        ;Set high order word
ADJ04:  ONES_[AC-OP]
ADJ06:  [K -1.0]BAR&[W5]_Q,              ;Copy P
         IF CT [MZ IX_MX] THEN [ADJBP9] ;If S=0 done
ADJBP1: Q-[W6]_F,F_Q,
         IF NOT CT [IN] THEN [ADJBP1]
;Count bytes in word
        J[44]_[W1]
        ONES_[W3],IX_MX          ;Count bytes in word here
ADJBP2: Q+[W6]_F,F_Q
        [W1]-Q_F,F_Y,
         IF CT [IN] THEN [ADJBP3]
        [W3]+1_[W3],IX_MX,               ;Count bytes in word
         GOTO [ADJBP2]
ADJBP3: [AC-OP]_[AC-OP+1],               ;Copy # of bytes to adjust
         IF CT [(MNxMOVR)!MZ IX_MX] THEN [SET NO DIVIDE]
        [W3]_[MEM-OP],IX_MX,             ;Copy divisor
         CALL [DIVSUB]                   ;Get quotient in MEM-OP
        [MEM-OP+1]_Y,                    ;Test remainder
         IF CT [IZ IX_MX] THEN [ADJBP5]
ADJBP5: ROR [W5]_[AC-OP],PUSH J[4]_CTR
        ROR [AC-OP]_[AC-OP],             ;Justify pointer in AC-OP
         RFCT
        J[4000]_[W1]
        SWAP [W1]_[W1],SEL AC+[0]
        [W1]&[AC-OP]_F,
         IF NOT CT [IZ] THEN [ADJBP6]
        [MEM-OP]+[AC-OP RH]_B,
         GOTOP [AC TO AC]
ADJBP6: CALL [READ NEXT]                 ;Get 2nd half of pointer
        [MB]+[MEM-OP]_B,SEL AC+[1]
        [K 7777.-1]&[MEM-OP]_B
        [K 7777.-1]BAR&[MB]_B
        [MEM-OP].OR.[MB]_Y,
         ALU_RAMFILE,SEL AC+[0]
        [AC-OP]_RAMFILE,SET LOCAL,
         GOTOP [IFETCH]

ADJBP9: SEL AC+[1],
         IF NOT CT [UZ] D [DISPATCH 4]  ;Dispatch if not global
        [MEM-OP+1]_RAMFILE,
         DISPATCH [DISPATCH 4]

;Here for IBP or ADJBP
ADJBP:  IF [AC.EQ.0] THEN [IBP0]
        ;GOTO [P&S SETUP]
```

```
.TOC    "       Byte Instructions"

;       +------+------+-+-+----+----------------+
;       !  P   !  S   !X!!!  AC!      Y          !
;       +------+------+-+-+----+----------------+
;        0    0 0    1 1 1 1  1 1               3
;        0    5 6    1 2 3 4  7 8               5
;Common setup routine for byte instructions
; Returns       W5/P    W6/S    Q/S     UZ set if global format

.IF/FTBYTE
P&S SETUP:
        2*[MEM-OP]_[W5],                ;Copy pointer to W5
         J[1]_CTR
P&S 2:  4*[W5]_B,                       ;Shift left 2 at a time
         LOOP [P&S 2]
        2*[W5]_Q                        ;Leave S left justified in Q
P&S 3:  ROL [MEM-OP]_[W5] ROL Q_Q,      ;Rotate pointer in W5, also rot Q
         J[4]_CTR
P&S 4:  ROL [W5]_[W5] ROL Q_Q,          ;Rotate pointer in W5, also rot Q
         LOOP [P&S 4]
        [K 77]&[W5]_B                   ;Mask P to 6 bits
        Q&[K 77]_F,F_Q_[W6],            ;Mask S to 6 bits
         IF [SECTION 0] THEN [P&S 5]
        [BIT12]&[MEM-OP]_Y,             ;Check for local/global format
         IF CT [IZ] THEN [P&S 5]
        [E]+1_F,F_Q_[E RH],1_UZ,        ;Make adr of 2nd half
         IF [LOCAL] THEN [P&S 6]
        Q_[E],GOTO [P&S 6]              ;Propagate carry
;Here if EFA calc is from first word
P&S 5:  [MEM-OP]_Y,SPEC SEL/XR,         ;Set AC and @ flags
         0_UZ,J[EFA CALC]_CTR           ; Load 2910 reg for subroutine call
;Here to do EFA calc for byte pointer
P&S 6:  IF CT [UZ] JSRP [FETCH IW]      ;Do EFA calc
        IF [NOT PXCT] THEN [P&S 64]
        J[0100]_[PXCT RH]
        CALL [PXCT FETCH]
        GOTO [P&S 66]
P&S 64: READ [E]                        ;Get word to write
P&S 66: J[BYTE MASK]_[W1]               ;Base of mask for byte size
        [W1]+[W6]_Y,Y_RAMFILE ADR       ;Address mask for byte
P&S 7:  [K -1]+[W5]_Y,ALU_CTR,IX_MX     ;Check P for 0 & load ctr
        RAMFILE_[W2],SEL AC+[0]         ;Get mask
         DISPATCH [DISPATCH 4]
```

```
;Setup routine for ILDB, IDPB, and IBP instructions
; Returns       W5/P    W6/S    Q/S     UZ set if global format

P&S SETUP:
        [BIT4]&[PC FLAGS]_Y,            ;Check first part done
         IF NOT CT [IZ] THEN [P&S SETUP]
IBPO:   2*[MEM-OP]_[W5],                ;Copy pointer to W5
         J[1]_CTR
IP&S 2: 4*[W5]_B,LOOP [IP&S 2]          ;Shift left 2 at a time
        [W5]+[W5]_Q                     ;Put S left justified in Q
        [K 7777.-1]BAR&Q_F,F_Q_[W6]     ;Leave only S in Q
        [MEM-OP]-Q_F,F_[MEM-OP],        ;Increment pointer
         IF CT [IC] THEN [IP&S 3]
        JS[4]_[W1]                      ;Put 400000,,4
        [K 7777.-1]&[MEM-OP]_B,         ;Clear old P
         PUSH J[2]_CTR
        ROR [W1]_[W1],RFCT              ;Leave 440000,,0 in W1
        [W1].OR.[MEM-OP]_B              ;Put 36. in P
        [MEM-OP]-Q_F,F_[MEM-OP],        ;Make pointer
         IF [SECTION 0] THEN [IP&S27]
        [BIT12]&[MEM-OP]_Y,
         IF CT [IZ] THEN [IP&S27]
        [E]+1_F,F_Q_[E RH],
         IF [LOCAL] THEN [IP&S24]
        Q_[E]                           ;Propagate carry
IP&S24: READ [E]                        ;Fetch 2nd half of pointer
        [K 7777.-1]BAR&[MB]_Q           ;Save non Y portion
        [MB]+1_[MB]                     ;Increment Y
        [K 7777.-1]&[MB]_B              ;Strip non Y portion
        Q.OR.[MB]_[MB],                 ;Add non Y portion
         CALL [MEMORY WRITE]
        ;[MB]+1_[MB],                   ;Increment 2nd half of pointer
        ; CALL [MEMORY WRITE]
        [K -1]+[E RH]_F,F_Q_[E RH],     ;Make adr of 1st half of pointer
         IF [LOCAL] THEN [IP&S 3]
        Q_[E],GOTO [IP&S 3]             ;Propagate carry
IP&S27: [MEM-OP]+1_[MEM-OP RH]          ;Increment adr
IP&S 3: [MEM-OP]_[MB],                  ;Write updated pointer in memory
         CALL [MEMORY WRITE]
        [W6]_Q,                         ;Load Q with S left justified
         DISPATCH [DISPATCH 3]          ;IBP is done, otherwise continue
IP&S 5: [BIT4].OR.[PC FLAGS]_B,         ;Set first part done
         GOTO [P&S 3]                   ; Rest like P&S setup
```

```
;Here for LDB or ILDB after P&S SETUP, memory word in MB
LDB:    IF CT [MN] THEN [LDB5]          ;Branch if P=0
        J[18.]_[W1]                     ;Half word shift
        [W5]-[W1]_Y,
         IF CT [IN IX_MX] THEN [LDB4]
        SWAP [MB]_[MB]                  ;Do first 18 shifts
        ZERO_[MB LH],                   ;Clear half
         IF CT [MZ] THEN [LDB5]
        [W5]-[W1]-1_F,F_Y,ALU_CTR       ;Load ctr with excess 18.
LDB4:   LRS [MB]_[MB],LOOP [LDB4]
LDB5:   [MB]&[W2]_Y,ALU_RAMFILE,        ;Mask to byte size
         GOTOP [TO NOWHERE CFPD]
```

```
;Here for DPB or IDPB after P&S SETUP, memory word in MB
DPB:    [W2]&[AC-OP]_B,                 ;Clear extra bits from byte
         IF CT [MN] THEN [DPB5]         ; Branch if P=0
        J[18.]_[W1]                     ;Half word shift
        [W5]-[W1]_Y,
         IF CT [IN IX_MX] THEN [DPB3]
        SWAP [AC-OP]_[AC-OP]            ;Do first 18 shifts
        ZERO_[AC-OP RH],                ;Clear half
         CALL [SWAP W2_W2]              ;Shift mask also
        ZERO_[W2 RH],                   ;Clear half of mask
         IF CT [MZ] THEN [DPb_.
        [W5]-[W1]-1_F,F_Y,ALU_CTR       ;Load ctr with excess 18.
DPB3:   [W2]_Q                          ;Put mask in Q reg
DPB4:   LLS [AC-OP]_[AC-OP] LLS Q_Q,
         LOOP [DPB4]
        Q_[W2]                          ;Get mask back from Q
DPB5:   [W2]BAR&[MB]_B,                 ;Clear old bits from mem word
         IF [NOT PXCT] THEN [DPB6]
        [AC-OP].OR.[MB]_B,              ;Set new bits
         CALL [PXCT STORE]              ;Write word back
        GOTO [TO NOWHERE CFPD]
DPB6:   [AC-OP].OR.[MB]_B,              ;Set new bits
         CALL [MEMORY WRITE]            ;Write word in memory
        ;GOTO [TO NOWHERE CFPD]
.ENDIF/FTBYTE
;Here when instruction done, clear first part done
TO NOWHERE CFPD:
        [BIT4]BAR&[PC FLAGS]_B,         ;Clear 1st part done and exit
         SET LOCAL,GOTOP [IFETCH]
```

```
.TOC    "       Floating Point"

;Format for floating point numbers is
; 0 1 2 3 4 5 6 7 8 9101112131415161718192021222324252627282930313233343 5
; ! \             / \                                                   /
; !       !                                 + --- fraction
; !       + --- exponent (excess 200)
; + --- sign

;Here for FIX instruction
FIX:
.IF/FTFP
        2*[MEM-OP]_[MEM-OP],
         IF NOT CT [MN] THEN [FIX 1]
        NEGATE [MEM-OP]_F,F_[MEM-OP]            ;Work with positive copy
FIX 1:  ROL [MEM-OP]_[MEM-OP],          ;Copy operand to extract exponent
         PUSH J[6]_CTR
        ROL [MEM-OP]_[MEM-OP],          ;Rotate exponent to rh of W1
         RFCT,REP/1
        [K 777]&[MEM-OP]_Q,             ;Get the exponent
         SET LOCAL
        [K 777]BAR&[MEM-OP]_B           ;Strip exponent from number
        J[242]_[W2]
        [W2]-Q_F,F_Y,ALU_CTR,IX_UX
        IF CT [UN] THEN [SET OVIT1 TRAP]
        [K -1]&[MEM-OP]_F,              ;Position result
         LSRC_B,O_SION SIOO_QION,
         IF CT [UZ] THEN [FIX 6]
FIX 5:  [K -1]&[MEM-OP]_F,REP/1,        ;Position result
         LSRC_B,O_SION SIOO_QION,
         LOOP [FIX 5]
FIX 6:  [MEM-OP]_RAMFILE,
         IF NOT CT [MN] D [OPERAND STORE]
        NEGATE [MEM-OP]_F,F_[MEM-OP],
         ALU_RAMFILE,
         DISPATCH [OPERAND STORE]
.ENDIF/FTFP

;Here for FIXR = fix and round
FIXR:
.IF/FTFP
        Q_Y,
         IF NOT CT [IN] THENP [IFETCH]
        [MEM-OP]+1_Y,ALU_RAMFILE,
         IF NOT CT [MN] THENP [IFETCH]
        [K -1]+[MEM-OP]_Y,ALU_RAMFILE,
         GOTOP [IFETCH]
        GOTO [TO AC]
.ENDIF/FTFP
```

```
;Here for FLTR = float and round
FLTR:    ;DISPATCH [OPERAND STORE]
```

```
;Here for FADx
FADX:   ;DISPATCH [OPERAND STORE]

;Here for FADRx
FADRX:  ;DISPATCH [OPERAND STORE]

;Here for FSBx
FSBX:   ;DISPATCH [OPERAND STORE]

;Here for FSBRx
FSBRX:  ;DISPATCH [OPERAND STORE]

;Here for FMPx
FMPX:   ;DISPATCH [OPERAND STORE]

;Here for FMPRx
FMPRX:  ;DISPATCH [OPERAND STORE]

;Here for FDVx
FDVX:   ;DISPATCH [OPERAND STORE]

;Here for FDVRx
FDVRX:  DISPATCH [OPERAND STORE]
```

```
.TOC    "       Full Word Data Transmission"

;Here for MOVEI
.IF/FAST
MOVEI:  [K -1.0]BAR&[E]_F,F_Y,          ;Strip extra bits
         ALU_RAMFILE,SET LOCAL,
         GOTOP [IFETCH]

;Here for MOVSI
MOVSI:  SWAP [E]_[E],SEL AC+[0]         ;Position bits
        [K -1.0]&[E]_Y,ALU_RAMFILE,
         SET LOCAL,GOTO [IFETCH]
.ENDIF/FAST

;Here for MOVSx (also some Hxx instructions)
MOVSX:  SWAP [MEM-OP]_[MEM-OP],
         DISPATCH [OPERAND STORE]

;Here for MOVMx
MOVMX:  IF NOT CT [MN] D [OPERAND STORE]
        ;GOTO [MOVNX]

;Here for MOVNx
MOVNX:  NEGATE [MEM-OP]_F,F_[MEM-OP],   ;Negate operand
         IF CT [IZ IX_MX] THEN [SET CO!CI STORE]
        IF NOT CT [MOVR] D [OPERAND STORE]
SET OV!CI!TI STORE:
        J[5002]_[WI],SET TRAP           ;Set overflow, carry 1, and trap 1
        SWAP [WI]_[WI],
         CALL [64*WI!PCF]
        SEL AC+[0],
         DISPATCH [OPERAND STORE]
```

```
.TOC    "       Fixed Point"

;Here for ADDx
ADDX:   [AC-OP]+[MEM-OP]_B,             ;Add AC & MEM
         IF CT [IOVR IX_MX] THEN [ADDXOV]
ADDX2:  IF CT [NOT MC] D [OPERAND STORE] ;No carry means done
SET CO!C1 STORE:
        [CO!C1].OR.[PC FLAGS]_B,
         SEL AC+[O],
         DISPATCH [OPERAND STORE]

;Here for SUBx
SUBX:   [AC-OP]-[MEM-OP]_B,             ;AC-MEM_MEM and then store
         IF CT [NOT IOVR IX_MX] THEN [ADDX2]

;Here if got overflow
ADDXOV:  IF CT [NOT MC] THEN [SET OV!C1!T1 STORE]
SET OV!CO!T1 STORE:
        J[6002]_[W1],SET TRAP           ;Set overflow, carry 0, and trap 1
        SWAP [W1]_[W1],
         CALL [64*W1!PCF]
        SEL AC+[O],
         DISPATCH [OPERAND STORE]
```

```
;Here for MULx or IMULx
; Memory operand already in Q
.IF 'FAST
MULX:   ZERO_[MEM-OP],J[33.]_CTR        ;Initialize work register
        TCM [AC-OP] [MEM-OP]            ;Do shift and add
MULX 2: TCM [AC-OP] [MEM-OP],REP/1,     ;Do shift and add
         LOOP [MULX 2]
.ENDIF/FAST
.IFNOT/FAST
MULX:   ZERO_[MEM-OP],J[34.]_CTR        ;Initialize work register
MULX 2: TCM [AC-OP] [MEM-OP],           ;Do shift and add
         LOOP [MULX 2]
.ENDIF/FAST
        TCM COR [AC-OP] [MEM-OP]        ;Final shift and add
        [MEM-OP]_F,B SEL/MEM-OP,ALSC_B, ;Shift product left
         SION_MC QION_SIOO O_QIOO,IX_MX ; put carry out in MC
        Q_F,(MN F)RS_[MEM-OP+1],        ;Get low order portion
         IF CT [MN] D [OPERAND STORE]
        IF NOT CT [MC] D [OPERAND STORE]
        BITO_[MEM-OP],                  ;Architecture sayes to do this
         CALL [SET OV!T1]               ;Set overflow & trap 1
        BITO_[MEM-OP+1],                ;More architecture cyberkrud
         DISPATCH [OPERAND STORE]

;Here for IMULx after doing MULx
IMULX:  [MEM-OP]+1_[MEM-OP],            ;In case high order portion neg
         IF CT [MZ IX_MX] THEN [IMULX4]
        IF NOT CT [MZ] CALL [SET OV!T1] ;Set overflow if high order bits
IMULX4: [MEM-OP+1]_[MEM-OP],
         DISPATCH [DISPATCH 4]
```

```
;Here for IDIVx
IDIVX:  [AC-OP]_[AC-OP+1],              ;Put argument in right place
         IF CT [IN] THEN [IDIVXN]
        ZERO_[AC-OP],GOTO [DIVXO]       ;Extend sign
IDIVXN: ONES_[AC-OP],GOTO [DIVXO]       ;Extend sign

;Here for DIVx
DIVX:   SEL AC+[1]                      ;Address low order portion
        RAMFILE_[AC-OP+1],NRFRD/1
DIVXO:  [MEM-OP]_[W3],IX_MX,            ;Copy divisor
         CALL [DIVSUB]
        DISPATCH [OPERAND STORE]

;Subroutine to do a divide
; Call with dividend in AC-OP, AC-OP+1 divisor in MEM-OP_W3,IX_MX
DIVSUB: [AC-OP]_[W2],                   ;Copy highorder dividend
         IF NOT CT [MN IX_MX] THEN [DIVXO2]
        NEGATE [W3]_F,F_[W3]            ;Make magnitude of divisor
DIVXO2: [AC-OP+1]+[AC-OP+1]_F,F_Q,      ;Put loworder dividend in Q
         IF NOT CT [MN] THEN [DIVXO4]
;Following 2 ins for KL10 compatible divide
        NEGATE Q_F,IX_UX                ;Latch carry out of loworder
        [W2]_BAR+C_F,UC_C,F_[W2]
;Following for non KL10 compatible
        ;[W2]BAR+C_F,O_CO,F_[W2]
DIVXO4: [W2]-[W3]_F,
         IF NOT CT [IN] THEN [SET NO DIVIDE]
        [AC-OP]_F,ASRC_B,B SEL/AC-OP,
         MN_SION SIOO_QION
.IF/FAST
        R SEL/MEM-OP,S SEL[AC-OP],
         O_CO,ALU SP FUN/DLN,ALU_Y/YES,
         SION_QIOO QION_SIOO,J[33.]_CTR
        TCDIV [MEM-OP] [AC-OP]_B
DIVX15: TCDIV [MEM-OP] [AC-OP]_B,REP/1,
         LOOP [DIVX15]
.ENDIF/FAST
.IFNOT/FAST
        R SEL/MEM-OP,S SEL[AC-OP],
         O_CO,ALU SP FUN/DLN,ALU_Y/YES,
         SION_QIOO QION_SIOO,J[34.]_CTR
DIVX15: TCDIV [MEM-OP] [AC-OP]_B,
         LOOP [DIVX15]
.ENDIF/FAST
        TCDIV COR [MEM-OP] [AC-OP]_B
        [AC-OP]_[MEM-OP+1],             ;Latch remainder sign
         IF CT [MN IX_UX] THEN [DIVX40] ; Branch if dividend negative
        [MEM-OP]_Y,                     ;Latch sign of divisor
         IF NOT CT [UN IX_MX] THEN [DIVX90] ;Positive remainder = OK
        IF CT [MN] THEN [DIVXR]
DIVXQ:  [MEM-OP]+[MEM-OP+1]_B           ;Add divisor to remainder
DIVXQ1: Q+[K -1]_[MEM-OP],              ;Adjust quotient and store
         RETURN

;Here because dividend was negative
DIVX40: [MEM-OP]_Y,                     ;Latch sign of divisor
         IF CT [UZ IX_MX] THEN [DIVX90] ;Remainder=0 OK
        [MEM-OP+1]+[W3]_Y,              ;Remainder vs divisor magnitude
```

```
        IF CT [UN IX_UX] THEN [DIVX46] ;R=negative, check quotient
DIVX42: IF CT [MN] THEN [DIVXQ]        ;Branch if divisor was negative
DIVXR:  [MEM-OP+1]-[MEM-OP]_A          ;Subtract divisor from remainder
        Q+1_F,F_[MEM-OP],              ;Adjust quotient and store
         RETURN
DIVX46: IF NOT CT [UZ] THEN [DIVX90]
        IF CT [MN] THEN [DIVXR]        ;Branch if divisor was negative
        [MEM-OP]+[MEM-OP+1]_B,
         GOTO [DIVXQ1]

DIVX90: Q_[MEM-OP],                    ;Save quotient
         RETURN

SET NO DIVIDE:
        J[40]_[W1]                     ;Will become no divide
        SWAP [W1]_[W1],
         CALL [W1!PCF]
SET OVIT1 TRAP:
        CALL [SET OVIT1],SET LOCAL     ;Set overflow and trap1
        GOTOP [TRAP]
```

```
.TOC    "       Double Precision Fixed Point"

;Here for DADD
DADD:   2*[AC-OP+1]_[AC-OP+1],          ;Adjust low order AC operand
         CALL [DADD4]                   ; and adjust low order mem operand
        [AC-OP+1]+[MEM-OP+1]_B,         ;Add low order parts
         IX_MX
        [AC-OP]+[MEM-OP]+C_B,           ;Add AC & MEM
         MC_C IX_MX
DADD2:  ASR [MEM-OP+1]_[MEM-OP+1],      ;Put sign on low order portion
         IF CT [MOVR] THEN [ADDXOV]
        IF CT [NOT MC] D [OPERAND STORE] ;No carry means done
        [CO!C1].OR.[PC FLAGS]_B,
         DISPATCH [OPERAND STORE]

DADD4:  2*[MEM-OP+1]_[MEM-OP+1],        ;Adjust low order mem operand
         RETURN

;Here for DSUB
DSUB:   2*[AC-OP+1]_[AC-OP+1],          ;Adjust low order AC operand
         CALL [DADD4]                   ; and adjust low order mem operand
        [AC-OP+1]-[MEM-OP+1]_B,         ;Subtract low order parts
         IX_MX
        [AC-OP]-[MEM-OP]-1+C_F,F_B,     ;AC-MEM_MEM and then store
         MC_C IX_MX,
         GOTO [DADD2]

;Here for DMUL
DMUL:   ;DISPATCH [OPERAND STORE]

;Here for DDIV
DDIV:   DISPATCH [OPERAND STORE]
```

```
.TOC    "       Shift and Rotate"

ASHC SETUP:
        [MEM-OP]_F,IX_MX                ;Latch MN
        2*[MEM-OP+1]_Q,                 ;Load Q with low order portion
         GOTO [ASH SETUP]

LSHC SETUP:
        [MEM-OP+1]_Q,                   ;Load Q with low order portion
          GOTO [ASH SETUP]

;Here for ASH, etc. setup
ASH SETUP:
        J[377]_[W1],SET LOCAL           ;Generate mask for extraneous bits
        [W1]&[E]_B,IX_UX,               ;Strip extraneous bits
         IF [B BIT 18] THEN [ASHR SETUP]
        [K -1]+[E]_B,ALU_CTR            ;Load counter
        SEL AC+[0],
         IF NOT CT [UZ] D [DISPATCH 3]
        SET LOCAL,GOTOP [IFETCH]
ASHR SETUP:
        [W1]-[E]_B,ALU_CTR              ;Load counter
        SEL AC+[0],
         DISPATCH [DISPATCH 4]

;Here for ASH
ASH:    [MEM-OP]+[MEM-OP]_B,            ;Shift left
         IX!UOVR_UX,LOOP [ASH]          ; Latch overflow if any
        [MEM-OP]_RAMFILE,SET LOCAL,     ;Write AC
         IF NOT CT [UOVR] THENP [IFETCH]
        [MEM-OP]+[MEM-OP]_B,            ;Shift off sign
         CALL [SET OV!T1]
        B[MEM-OP]_F,MN_SIGN SIGN_QIGN,
         LRS F_B,ALU_RAMFILE,
         GOTOP [TRAP]

ASHR:   ASR [MEM-OP]_[MEM-OP],          ;Shift right
         LOOP [ASHR]
        [MEM-OP]_RAMFILE,               ;Write AC
         SET LOCAL,
         GOTOP [IFETCH]

;Here for ASHC
ASHC:   ASRC [MEM-OP]_[MEM-OP]          ;Shift right once
ASCH1:  R SEL/MEM-OP,S SEL[MEM-OP],     ;Shift left & latch overflow
         IX!UOVR_UX,
         Q_CO,ALU SP FUN/DLN,ALU_Y/YES,
         QIGN_SIGN Q_QIGN,LOOP [ASCH1]
        LLSC [MEM-OP]_[MEM-OP]          ;Shift left
        LLSC [MEM-OP]_[MEM-OP]          ;Shift left
        MN_SIGN SIGN_QIGN,LSRC_B,       ;Shift right adding sign
         B[MEM-OP]_F,ALU_RAMFILE        ; Save AC
        SEL AC+[1]                      ;Address AC+1
        Q_F,LRS F_B,MN_SIGN SIGN_QIGN,
         B SEL/MEM-OP+1,ALU_RAMFILE,
         IF NOT CT [UOVR] THEN [IFETCH]
        GOTOP [SET OV!T1 TRAP]

;Here for an ASH shifting right
```

```
ASHCR:  ASRC [MEM-OP]_[MEM-OP],          ;Shift right
         LOOP [ASHCR]
        [MEM-OP]_Y,ALU_RAMFILE,
         SEL AC+[1]
        Q_F,LRS F_B,MN_SION SIOC_QION,
         B SEL/MEM-OP+1,
         ALU_RAMFILE,
         GOTOP [IFETCH]
```

```
;Here for LSH
LSH:    2*[MEM-OP]_[MEM-OP],
         LOOP [LSH]
        [MEM-OP]_RAMFILE,               ;Write AC
         SET LOCAL,
         GOTOP [IFETCH]
LSHR:   LRS [MEM-OP]_[MEM-OP],          ;Shift right
         LOOP [LSHR]
        [MEM-OP]_RAMFILE,               ;Write AC
         SET LOCAL,
         GOTOP [IFETCH]

;Here for LSHC
LSHC:   LLSC [MEM-OP]_[MEM-OP],         ;Shift left with Q
         LOOP [LSHC]
        Q_[MEM-OP+1].
         GOTO [D TO AC.O]
LSHCR:  LSRC [MEM-OP]_[MEM-OP],         ;Shift right with Q
         LOOP [LSHCR]
        Q_[MEM-OP+1],
         GOTO [D TO AC.O]

;Here for ROT
ROT:    ROL [MEM-OP]_[MEM-OP],
         LOOP [ROT]
        [MEM-OP]_RAMFILE,               ;Write AC
         SET LOCAL,
         GOTOP [IFETCH]
ROTR:   ROR [MEM-OP]_[MEM-OP],
         LOOP [ROTR]
        [MEM-OP]_RAMFILE,               ;Write AC
         SET LOCAL,
         GOTOP [IFETCH]

;Here for ROTC
ROTC:   ROLC [MEM-OP]_[MEM-OP],         ;Rotate left
         LOOP [ROTC]
        Q_[MEM-OP+1],
         GOTO [D TO AC.O]
ROTCR:  RORC [MEM-OP]_[MEM-OP],         ;Rotate right
         LOOP [ROTCR]
        Q_[MEM-OP+1],
         GOTO [D TO AC.O]
```

```
;Register usage:
;       MEM-OP  AC contents
;       AC-OP   BLT limit
;Here for BLT
BLT:    [K 7777.-1]&[E]_B               ;Mask E to 30 bits
        [E]_[AC-OP],                    ;Copy final adr
         IF [PXCT] THEN [BLT PXCT]
.IF/FAST
        SWAP [MEM-OP]_[E RH]            ;Make first address to read
        [E]_[W1]                        ;Copy section number
        [MEM-OP]_[W1 RH]                ;Make destination adr
        [W1]-[E]-1_F,F_Y,               ;Check for "clear core " case
         IF CT [IZ] THEN [BLT+1]
.ENDIF/FAST
BLT 2:  SWAP [MEM-OP]_[E RH],           ;Make adr of location to read
         CALL [MEM READ 0]              ;Get word to transfer
        [MEM-OP]_[E RH],                ;Make destination adr
         CALL [MEMORY WRITE]
        [E]-[AC-OP]_Y,SEL AC+[0],IX_MX
        [BIT17]+[MEM-OP]+1_F,F_B,       ;Increment pointer
         IF NOT CT [MN] THEN [TO NOWHERE]
        [MEM-OP]_Y,ALU_RAMFILE,         ;Remember how far we got
         GOTO [BLT 2]

BLT PXCT:
        SWAP [MEM-OP]_[E RH],           ;Make adr of loc to read
         CALL [PXCT FETCH 40]
        [MEM-OP]_[E RH],                ;Make destination adr
         CALL [PXCT STORE 200]          ; Copy data
        [E]-[AC-OP]_Y,SEL AC+[0],IX_MX
        [BIT17]+[MEM-OP]+1_F,F_B,       ;Increment pointer
         IF NOT CT [MN] THEN [TO NOWHERE]
        [MEM-OP]_Y,ALU_RAMFILE,         ;Remember how far we got
         GOTO [BLT 2]

.IF/FAST
;Here for BLT [#,#+1]
BLT+1:  READ [E]                        ;Fetch first word in block
        [MEM-OP]_[E RH],                ;Write next word
         CALL [MEMORY WRITE]
        GOTO [BLT 14]
BLT+1N: [BIT17]+[MEM-OP]+1_F,F_B,
         ALU_RAMFILE,
         CHECK INTERRUPTS
        [E]+1_F,F_Q_[E],
         SPEC SEL/PAGE TABLE ENTRY,
         CALL [MEM WRITE 1]
BLT 14: [AC-OP]-[E]-1_F,F_Y,
         SEL AC+[0],
         IF NOT CT [IN] D [DISPATCH 3]
        SET LOCAL,GOTO [IFETCH]
.ENDIF/FAST
```

```
.TOC    "       Program Control"

;Here for AOBJX instructions
AOBJX:  [MFM-OP]+1_[MEM-OP RH]          ;Increment RH of AC
        [BIT17]+[MEM-OP]_B,IX_MX,       ;Increment LH
         DISPATCH [DISPATCH 3]
```

```
;Here for JRST
JRST:   2*[IR]_[W1],
         IF [AC.EQ.0] THEN [JXA]
        4*[W1]_B,                       ;Keep shifting
         CALL [2*W1_W1 LOAD IR]
        [K 7777.-1]&[PC]_B,             ;Reduce PC to 30 bits
         SECTION SELECT,                ;Load SECTION 0 from PC
         DISPATCH [MISC]

;Jump and restore flags from bits 0-12 of the final indirect or index word
; JRST 2,
JRSTF:
.IF/SILLY
        [IR]_[MB],                      ;Reseed MB
         CALL [EFA CALC]
.ENDIF/SILLY
        IF [NOT SECTION 0] THEN [MUUO]
        [E]_[PC RH]                     ;New PC
JRSTFX: LRS [BIT4]_[W1],                ;Make a user mode bit
         IF [EXEC] THEN [MB_PC FLAGS]
        [W1].OR.[MB]_B                  ;Will still be user mode
        LRS [W1]_[W2]                   ;Make a user I/O bit
        [PC FLAGS]BAR&[W2]_B            ;Make mask for user I/O
        [W2]BAR&[MB]_B                  ;Prohibit new user I/O
MB_PC FLAGS:
        [MB]+[MB]_Q,                    ;Make user into bit4
         J[23B MASK]_RAMFILE ADR
        Q&[BIT4]_Y,IX_MX,SET EXEC,      ;Latch new exec as MZ
         IF [TOPS10] CALL [ZERO_PC LH]  ;Size PC for TOPS10
        [MB]_Q RAMFILE_[W2],            ;Get mask for allowable flags to set
         IF NOT CT [MZ] CALL [SET USER LATCH]
        [W2]BAR&Q_F,F_[PC FLAGS]        ;Set new flags
         SET LOCAL,
         GOTOP [TRAP]

SET USER LATCH:
        SET USER,RETURN

;JRST 4,
HALT:   IF [USER] CALL [CHECK IO OK]    ;Check if this is legal
        J[HALT INS]_[W1]                ;Code for JRST 4,
        [E]_[PC],                       ;Set new PC from E
         GOTOP [SET HALT CODE]

;Here for a JRST 5,
XJRSTF:
        READ [E]                        ;Get new flag word
        [MB]_[MEM-OP],                  ;Save flag word
         CALL [READ NEXT]               ; Get new PC
        [MB]_[PC],                      ;Set new PC
         IF [TOPS10] CALL [ZERO_PC LH]
        [MEM-OP]_[MB],
         GOTOP [JRSTFX]

ZERO_PC LH:
        IF [NOT PAGED] RETURN           ;If not paged don't do
        ZERO_[PC LH],RETURN             ;Be sure only 18 bits
```

```
;JRST 6,
XJEN:   READ [E]                        ;Get new flag word
        [MB]_[MEM-OP],                  ;Save flag word
         CALL [READ NEXT]               ; Get new PC
JNO:    CALL [DISMISS INT]              ;Dismiss interrupt
                                        ; If not legal do an MUUO
        [MB]_[PC],                      ;Set new PC
         IF [TOPS10] CALL [ZERO_PC LH]
        [MEM-OP]_[MB],
         GOTO [MB_PC FLAGS]

;JRST 7,
XPCW:   IF [USER] CALL [CHECK IO OK]    ;Be sure legal
        [PC FLAGS]_[MB],                ;Save flags
         CALL [MEMORY WRITE]
        [PC]_[MB],
         CALL [WRITE NEXT]
        CALL [READ NEXT]                ;Get new flags
        [MB]_[MEM-OP],                  ;Save new flags
         CALL [READ NEXT]               ; Get new PC
        [MB]_[PC],                      ;Set new PC
         IF [TOPS10] CALL [ZERO_PC LH]
        [MEM-OP]_[MB],
         GOTO [MB_PC FLAGS]

;JRST 10,
JRST 10:
        CALL [DISMISS INT]
        [E]_[PC],
         SET LOCAL,
         GOTOP [IFETCH]
```

```
;JRST 12,
JEN:    [MB]_[MEM-OP],                      ;Save new flags
         IF [NOT SECTION 0] THEN [MUUO]     ;Not legal in section 0
        [E]_[MB],                           ;Save new PC
         GOTOP [XJNO]

;Here to dismiss an interrupt
; If IO not legal will do an MUUO
; uses Q, W1, W2, updates RF(PI IN PROG) and PI REG
DISMISS INT:
        J[PI IN PROG]_RAMFILE ADR           ;Address of Pi's in progress
        J[100]_[W2]                         ;Mask for highest PI
        [W2]_Q RAMFILE_[W2],IX_MX,          ;Put mask in Q, Pi's in W2
         IF [USER] CALL [CHECK IO OK]       ;Be sure IO is legal
        IF CT [MZ] RETURN                   ;Check if any Pi's in progress
JEN2:   Q&[W2]_F,F_Y RORQ_Q,
         IF CT [IZ] THEN [JEN2]
        Q_F,ROL F_B,B SEL/W1                ;Make mask for level
        [W1]BAR&[W2]_B,ALU_RAMFILE,         ;One less PI in progress
         IF NOT CT [IZ] THEN [JEN3]
        ZERO_[PI REG],Y_2914 STATUS,
         RETURN

JEN3:   Q&[W2]_F,F_Y RORQ_Q,IX_MX
        [K -1]+[PI REG]_B,
         Y_2914 STATUS,
         IF CT [MZ] THEN [JEN3]
        RETURN

SFM:    IF [NOT SECTION 0] THEN [SFM 2] ;Legal in nonzero sections
        IF [USER] CALL [CHECK IO OK]    ;Or if IO is legal
SFM 2:  [PC FLAGS]_[MEM-OP],            ;Copy flags and then write them
         GOTO [TO MEM]
```

```
;Here for JFFO
JFFO:   ZERO_[W1],                          ;Count leading zeros here
         IF CT [MZ] THEN [JFFO X]
        [E]_[PC],                           ;Take branch
         IF CT [MN] THEN [JFFO X]
JFFO 1: [MEM-OP]+[MEM-OP]_B,1X_MX           ;Left shift word
        [W1]+1_[W1],                        ;Count leading zeros
         IF NOT CT [MN] THEN [JFFO 1]
JFFO X: SEL AC+[1]                          ;Address AC+1
        [W1]_RAMFILE,                       ;Clear AC+1
         GOTOP [IFETCH]
```

```
;Here for JFCL
JFCL:   J[JFCL MASK]_RAMFILE ADR        ;Point to 037777 777777
        2*[IR]_[W1],                    ;Copy IR and shift left once
         J[3]_CTR
JFCL4:  4*[W1]_B,LOOP [JFCL4]           ;Shift copy of IR
        RAMFILE_[W2],                   ;Get mask for JFCL test bits
         CALL [W2BAR&W1_B]              ; Leave only bits to test
        [W1]&[PC FLAGS]_Y,              ;Test flag bits
         SET LOCAL,
         IF CT [IZ] THENP [IFETCH]
        [W1]BAR&[PC FLAGS]_B,           ;Clear flags
         GOTO [JXA]
```

```
;Here for XCT
;       MEM-OP contains instruction to execute
XCT:    [K -1.0]&[IR]_Q,                ;In case this is PXCT
         IF [USER] THEN [IFETCH 4]
        Q_[PXCT],                       ;In case this is PXCT
         IF [AC.EQ.0] THEN [IFETCH 4]   ;If plain just do it
        [MEM-OP]_[IR],SET PXCT,LOAD IR, ;Set flag this is PXCT
         GOTOP [IFETCH PXCT]

PXCT STORE 40:
        J[0040]_[PXCT RH]
        GOTO [PXCT STORE]

;Here for usual case of writing operands to memory
PXCT STORE 200:
        J[0200]_[PXCT RH]               ;Flavour of store
        ;GOTO [PXCT STORE]
;Here when doing memory write under PXCT
; Call J[testbit]_[PXCT RH]
PXCT STORE:
        CALL [SET PXCT CTXT]
        [K 7777.-1]&[E]_F,F_Q_B,        ;Get 1st operand
         SPEC SEL/PAGE TABLE ENTRY,
         CALL [MEMORY WRITE]
        GOTO [SET CURRENT CTXT]

PXCT FETCH 40:
        J[0040]_[PXCT RH]
        GOTO [PXCT FETCH]

;Here for usual PXCT FETCH (i.e. 200)
PXCT FETCH 200:
        J[0200]_[PXCT RH]               ;Flavour of fetch
        ;GOTO [PXCT FETCH]
;Here when doing a memory read under PXCT
; Call J[testbit]_[PXCT RH]
PXCT FETCH:
        CALL [SET PXCT CTXT]
        [K 7777.-1]&[E]_F,F_Q_B,        ;Get 1st operand
         SPEC SEL/PAGE TABLE ENTRY,
         CALL [MEMORY READ]
        ;GOTO [SET CURRENT CTXT]

;Here to set current context
SET CURRENT CTXT:
        J[PROC REG]_RAMFILE ADR         ;Get what AC blocks should be
        RAMFILE_[W2 LH],NRFRD/1,
         CLEAR USER
        J[RF PROC REG]_RAMFILE ADR      ;Get current proc reg
        RAMFILE_[W2 RH],NRFRD/1
        [W2]_Y,ALU_PROC REG             ;Set proper processor reg
        [W2]_RAMFILE,
         RETURN

SET PXCT CTXT:
        SWAP [PXCT]_[W1]                ;Copy test bits
        [W1]&[PXCT]_Y,
         IF CT [IZ] RETURN
```

```
        ;GOTO [SET PREV CTXT]
;Here to set previous context
; Called only in exec mode !!
        J[4000]_[W1]                    ;bit 24
        SWAP [W1]_[W1]                  ;Becomes bit 6=prev context user
        [W1]&[PC FLAGS]_Y,IX_MX         ;See if he can do this
        J[RF PROC REG]_RAMFILE ADR      ;Address last value of PROC REG
        RAMFILE_[W1],NRFRD/1,           ;Get last PROC REG
         IF NOT CT [MZ] CALL [SET USER LATCH]
        2*([W1]+[W1 LH])_B              ;Shift left twice
        2*[W1]_[W1 LH]                  ;Puts prev AC in position
        [W1]_RAMFILE                    ;In case clock interrupts
        [W1]_Y,ALU_PROC REG,
         RETURN
        ;RAMFILE_[E LH],                ;Get previous section
        ;J[PCS]_RAMFILE ADR             ;Remember previous section
```

```
;Here for MAP
MAP:    [K 7777.-i]&[E]_B,              ;Strip E to 30 bits
         SPEC SEL/PAGE TABLE ENTRY,     ; Set section flags
         IF [USER] CALL [CHECK IO OK]   ;Be sure allowed to do this
        J[MAP 3]_[W1]
        CALL [SET PF RCOVR]
        ZERO_[W1],0_UC,
         IF [PAGED] CALL [PAGE R REFIL] ;Get page info for E
MAP 3:  [K 777]&[E]_B,SEL AC+[0]        ;Leave only on page portion
        J[1000]_[W2]                    ;Will become V bit
        SWAP [W2]_[W2],                 ;Make bit8 = V bit
         IF [PAGED] CALL [W2.OR.W1]
        [E].OR.[W1]_B,ALU_RAMFILE,      ;Add on page portion
         SET LOCAL,GOTOP [IFETCH]
```

```
.TOC    "       Stack Operations"

;Here for ADJSP after FETCH AC&I and CHK PC SECT
ADJSP:  [K 407777.0]&[AC-OP]_Y,IX_MX,   ;Latch sign of stack pointer
         SEL AC+[0],                    ; Address AC
         IF [SECTION 0] D [DISPATCH 4]  ; D4 = ADJSP V
        SET LOCAL,
         IF CT [(MNxMOVR)!MZ] THEN [ADJSP V]
        [AC-OP]+[MEM-OP]_B,ALU_RAMFILE, ;Add adjustment
         IF [NOT B BIT 18] THEN [TO AC]
        [K -1.0]+[MEM-OP]_B,ALU_RAMFILE,
         GOTO [TO AC]

ADJSP V:
        [MEM-OP]+[AC-OP RH]_B,          ;Adjust RH
         SET LOCAL
        SWAP [MEM-OP]_[MEM-OP]          ;Prepare to add to LH
        [MEM-OP]+[AC-OP]_B,ALU_RAMFILE,
         IF NOT CT [INxorMN] THEN [IFETCH]
        [MEM-OP]_Y,                     ;Compare sign of E to AC
         IF CT [INxorMN] CALL [SET TRAP 2]
        GOTO [TRAP]
```

```
;Here for PUSHJ
; CHK PS SECT loaded SECTION 0 from PC and loaded MB with PC
PUSHJ:  [E]_[MEM-OP+1]                  ;Save adr where we will go
        [PC FLAGS]_Q,                   ;Save pc flags for sect0 case
         J[PUSHJ FLAGS]_RAMFILE ADR     ;Adr of mask for flags to clear
        RAMFILE_[W6],                   ;Get mask
         IF [SECTION 0] THEN [PUSHJ VANILLA]
        [K 407777.0]&[MEM-OP]_Y,        ;Check for which extended case
         IX_MX
        SWAP [K 777]_[W1]               ;9bit mask for PC section
                                        ; PC sect to 9 bits(strange but ok)
        [W1]&[MB LH]_B,                 ;We are going to write PC in memory
         IF CT [(MNxMOVR)!MZ] THEN [PJ H]
        [MEM-OP]+1_[MEM-OP],SET GLOBAL  ;Increment pointer
        WRITE [MEM-OP]
        [W6]BAR&[PC FLAGS]_B,           ;Clear flags
         SEL AC+[0]                     ;Select AC to restore pointer
        [MEM-OP+1]_[PC],
         GOTO [TO AC]

PUSHJ VANILLA:
        Q_[MB LH]                       ;<pc flags,,PC> in MB
PJ H:   [BIT17]+[MEM-OP LH]_B           ;Increment LH
        [MEM-OP]+1_[MEM-OP RH]          ;Increment RH of AC
        [MEM-OP]_[E RH],SET LOCAL,      ;Address to write into
         CALL [MEMORY WRITE]
        [K -1.0]&[MEM-OP]_Y,            ;Check for overflow
         IX_MX,SEL AC+[0]
        [W6]BAR&[PC FLAGS]_B            ;Clear flags
        [MEM-OP+1]_[PC],                ;Perform branch
         SET LOCAL,
         IF NOT CT [MZ] THENP [TO AC]
        [MEM-OP]_Y,ALU_RAMFILE,         ;Update pointer
         GOTO [PUSH TRAP]               ;Set trap flags

;Here to check PC section, set MZ if section 0
CHK PC SECT:
        [PC]_[MB],SECTION SELECT,       ;Select our section
         DISPATCH [DISPATCH 3]
```

```
;Here for PUSH
; CHK PC SECT loaded SECTION 0 from PC
PUSH:   [PC]_Y,SECTION SELECT           ;Select our section
        [K 407777.0]&[AC-OP]_Y,         ;Test for extended case
         IX_MX,
         IF [SECTION 0] THEN [PUSH VANILLA]
        IF CT [(MNxMOVR)!MZ] THEN [PUSH VANILLA]
        [AC-OP]+1_[E],                  ;New stack address
         IF [PXCT] CALL [PXCT STORE 40]
        [K 7777.-1]&[E]_F,F_Q_B,
         SPEC SEL/PAGE TABLE ENTRY,
         IF [NOT PXCT] CALL [MEM WRITE 1]
        SEL AC+[0]                      ;Address AC
        [AC-OP]+1_Y,ALU_RAMFILE,        ;Update AC
         CLEAR LOCAL,
         GOTOP [IFETCH]

PUSH VANILLA:
        [BIT17]+[AC-OP LH]_B            ;Increment LH of stack pointer
        [AC-OP]+1_[AC-OP RH]            ;Increment RH
        [AC-OP]_[E RH],                 ;Set in section address
         IF [PXCT] CALL [PXCT STORE 40]
        [K 7777.-1]&[E]_F,F_Q_B,        ; Data already in MB
         SPEC SEL/PAGE TABLE ENTRY,
         IF [NOT PXCT] CALL [MEM WRITE 1]
        [K -1.0]&[AC-OP]_Y,             ;Check for overflow
         IX_MX,SEL AC+[0]               ; Address AC to restore updated version
        [AC-OP]_RAMFILE,                ;Restore stack pointer
         SET LOCAL,
         IF NOT CT [MZ] THENP [IFETCH]
PUSH TRAP:
        CALL [SET TRAP 2]               ;Set trap 2 flag
        SET LOCAL,GOTOP [TRAP]
```

```
;Here for POP after CHK PC SECT
POP:    [K 407777.0]&[MEM-OP]_Y,IX_MX,  ;Test stack pointer
         IF [SECTION 0] THEN [POP VANILLA]
        [E]_[W6],                       ;Save effective address
         IF CT [(MNxMOVR)!MZ] THEN [POP V 1]
        [MEM-OP]_Q_[E],                 ;Address of stack data
         IF [GLOBAL] THEN [POP 3]
        [K 7777.-1]&[E]_B,CLEAR LOCAL,
         IF [PXCT] CALL [PXCT FETCH 40]
        [E]_Q_[E],
         SPEC SEL/PAGE TABLE ENTRY,
         IF [NOT PXCT] CALL [MEMORY READ]
        SET LOCAL,GOTO [POP 5]          ;Restore effective address
POP 3:  [K 7777.-1]&[E]_B,              ;Read location off stack
         IF [PXCT] CALL [PXCT FETCH 40]
        [E]_Q_[E],
         SPEC SEL/PAGE TABLE ENTRY,
         IF [NOT PXCT] CALL [MEMORY READ]       ;Read location off stack
POP 5:  [W6]_[E],
         IF [PXCT] CALL [PXCT STORE 200]
        [K 7777.-1]&[E]_F,F_Q_B,
         SPEC SEL/PAGE TABLE ENTRY,
         IF [NOT PXCT] CALL [MEM WRITE 1]
        SEL AC+[0]
        [K -1]+[MEM-OP]_Y,              ;Decrement stack pointer
         ALU_RAMFILE,
         SET LOCAL,
         GOTOP [IFETCH]                 ; Store stack pointer

POP V 3:
        [MEM-OP]_[E RH],SET LOCAL,      ;Get in section address
         IF [PXCT] CALL [PXCT FETCH 40]
        [K 7777.-1]&[E]_F,F_Q_B,        ; Get data from stack
         SPEC SEL/PAGE TABLE ENTRY,
         IF [NOT PXCT] CALL [MEMORY READ]
        SET GLOBAL,
         GOTO [POP V 5]

POP VANILLA:
        [E]_[W6]                        ;Save effective address
POP V 1:
        [PC]_[E],                       ;Copy pdl section number
         IF [GLOBAL] THEN [POP V 3]
        [MEM-OP]_[E RH],SET LOCAL,      ;Get in section address
         IF [PXCT] CALL [PXCT FETCH 40]
        [K 7777.-1]&[E]_F,F_Q_B,        ; Get data from stack
         SPEC SEL/PAGE TABLE ENTRY,
         IF [NOT PXCT] CALL [MEMORY READ]
POP V 5:
        [W6]_[E],
         IF [PXCT] CALL [PXCT STORE 200]
        [K 7777.-1]&[E]_F,F_Q_B,
         SPEC SEL/PAGE TABLE ENTRY,
         IF [NOT PXCT] CALL [MEM WRITE 1]
        ;GOTO [POP V PNTR]
POP V PNTR:
        [K -1.0]&[MEM-OP]_Y,            ;Check for overflow
         IX_MX
```

```
[MEM-OP]-[BIT17]_A,SEL AC+[0]    ;Decrement LH
[K -1]+[MEM-OP RH]_B,            ;Decrement RH of pointer
 SET LOCAL,
 IF NOT CT [MZ] THENP [TO AC]
[MEM-OP]_RAMFILE,                ;Store updated pointer
 GOTO [PUSH TRAP]                ; Then trap
```

```
;Here for POPJ
POPJ:
.IF,FTMBZ
        [K -1.0]BAR&[IR]_Y,             ;E should be 0
         IF NOT CT [IZ] THEN [MUUO]
.ENDIF/FTMBZ
        [K 407777.0]&[MEM-OP]_Y,        ;Test stack pointer
         IX_MX,
         IF [NOT SECTION 0] THEN [POPJ 1]
        [MEM-OP]_[E RH],                ;Fetch new PC
         CALL [MEM READ 0]
        [MB]_[PC RH],                   ;Set new PC
         GOTO [POP V PNTR]

POPJ 1: [MEM-OP]_[E RH],SET LOCAL,
         IF NOT CT [(MNxMOVR)!MZ] THEN [POPJ 2]
        READ [E]                        ;Fetch new PC
        [MB]_[PC],                      ;Set new PC
         GOTO [POP V PNTR]

POPJ 2: [MEM-OP]_[E],SET GLOBAL,        ;Stack address
         CALL [MEM READ 0]              ;Fetch new PC
        [MB]_[PC],                      ;Set new PC
         SEL AC+[0]
        [K -1]+[MEM-OP]_Y,              ;Save new AC
         ALU_RAMFILE,
         SET LOCAL,
         GOTOP [IFETCH]
```

```
;Here to put flags & PC in MB or PC in MB depending on section
; First part of JSR and JSP
JSR SETUP:
        J[PUSHJ FLAGS]_RAMFILE ADR
        RAMFILE_[W5],NRFRD/1,           ;Get bits to clear from flags
         IF [SECTION 0] THEN [JSR SECT 0]
        [K 7777.-1]&[MB]_B,             ;Mask off bits 0-5
         SEL AC+[0],
         DISPATCH [INST EXCT]
JSR SECT 0:
        [PC FLAGS]_[MB LH],SEL AC+[0],  ;Put flags in memory lh
         DISPATCH [INST EXCT]
```

```
;Here for JSR
JSR:
.IF/FTMBZ
        IF [AC.NE.0] THEN [MUUO]        ;AC field should be 0
.ENDIF/FTMBZ
        CALL [MEMORY WRITE]             ;MB already setup by JSR SETUP
        [E]+1_[E RH]                    ;Go to next location

JSR CLEAR FLAGS:
        [W5]BAR&[PC FLAGS]_B            ;Clear 1st part done, Address failure
                                        ; inhibit, and trap flags
        [E]_[PC],SET LOCAL,             ;Set new PC
         GOTOP [IFETCH]

;Here for JSP
JSP:    [MB]_RAMFILE,                   ;Write AC and exit
         GOTO [JSR CLEAR FLAGS]
```

```
;Here for JSA
JSA:    [MEM-OP]_[MB],                  ;Copy AC to write in E
         CALL [MEMORY WRITE]
        SWAP [E]_[MEM-OP]               ;Put E rh in mem lh
        [PC]_[MEM-OP RH]                ;Put PC rh in mem rh
        [E]_[PC]                        ;Set new PC
        [PC]+1_[PC RH],                 ;Make E+1 new PC
         GOTO [TO AC.0]

;Here for JRA
JRA:    [E]_[W6]                        ;Save E
        SWAP [MEM-OP]_[E RH],           ;Address to load AC with
         CALL [MEM READ 0]
        [W6]_[PC],SEL AC+[0]            ;Set new PC
        [MB]_Y,ALU_RAMFILE,             ;Write AC and done
         SET LOCAL,GOTO [IFETCH]
```

```
.TOC    "       Arithmetic Testing"

;Here for CAxx instructions to compare operands
CAX:    [AC-OP]-[MEM-OP]_Y,IX_MX,       ;Compare operands and dispatch
         DISPATCH [DISPATCH 3]

;Here for AOxx instructions to increment AC or memory
AOXX:   [MEM-OP]+1_[MEM-OP],            ;Increment and dispatch
         IF CT [IZ IX_MX] THEN [SET CO!C1 STORE]
        IF NOT CT [MOVR] D [DISPATCH 3]
        GOTO [SET OV!C1!T1 STORE]

;Here for SOxx instructions to decrement AC or memory
SOXX:   [K -1]+[MEM-OP]_B,              ;Decrement and dispatch
         IF CT [MZ IX_MX] D [DISPATCH 3]
        IF NOT CT [MOVR] THEN [SET CO!C1 STORE]
        GOTO [SET OV!CO!T1 STORE]
```

```
;Here for SxL instructions to conditionaly skip
SXL:    IF NOT CT [MNxMOVR] D [DISPATCH 4]
        [PC]+1_[PC RH],
         DISPATCH [DISPATCH 4]

;Here for SxE instructions to conditionaly skip
SXE:    IF NOT CT [MZ] D [DISPATCH 4]
        [PC]+1_[PC RH],
         DISPATCH [DISPATCH 4]

;Here for SxLE instructions to conditionaly skip
SXLE:   IF NOT CT [(MNxMOVR)!MZ] D [DISPATCH 4]
        ;GOTO [SXA]

;Here for SxA instructions to conditionaly skip
SXA:    [PC]+1_[PC RH],
         DISPATCH [DISPATCH 4]

;Here for SxGE instructions to conditionaly skip
SXGE:   IF CT [MNxMOVR] D [DISPATCH 4]
        [PC]+1_[PC RH],
         DISPATCH [DISPATCH 4]

;Here for SxN instructions to conditionaly skip
SXN:    IF CT [MZ] D [DISPATCH 4]
        [PC]+1_[PC RH],
         DISPATCH [DISPATCH 4]

;Here for SxG instructions to conditionaly skip
SXG:    IF CT [(MNxMOVR)!MZ] D [DISPATCH 4]
        [PC]+1_[PC RH],
         DISPATCH [DISPATCH 4]
```

```
;Here for JxL instructions to conditionaly branch
JXL:    IF NOT CT [MN] D [DISPATCH 4],
         SET LOCAL
        [E]_[PC],DISPATCH [DISPATCH 4]

;Here for JxE instructions to conditionaly branch
JXE:    IF NOT CT [MZ] D [DISPATCH 4],
         SET LOCAL
        [E]_[PC],DISPATCH [DISPATCH 4]

;Here for JxLE instructions to conditionaly branch
JXLE:   IF NOT CT [(MNxMOVR)!MZ] D [DISPATCH 4],
         SET LOCAL
        [E]_[PC],DISPATCH [DISPATCH 4]

;Here for JxA instructions to conditionaly branch
JXA:    [E]_[PC],SET LOCAL,
         DISPATCH [DISPATCH 4]

;Here for JxGE instructions to conditionaly branch
JXGE:   IF NOT CT [NOT MN] D [DISPATCH 4],
         SET LOCAL
        [E]_[PC],DISPATCH [DISPATCH 4]

;Here for JxN instructions to conditionaly branch
JXN:    IF CT [MZ] D [DISPATCH 4],
         SET LOCAL
        [E]_[PC],DISPATCH [DISPATCH 4]

;Here for JxG instructions to conditionaly branch
JXG:    IF CT [(MNxMOVR)!MZ] D [DISPATCH 4],
         SET LOCAL
        [E]_[PC],DISPATCH [DISPATCH 4]
```

```
;Here for SKIPx instructions to store in AC if AC.ne.0
SKIPX:  SET LOCAL,
         IF [AC.EQ.0] THENP [IFETCH]     ;If nc AC we're done
        [MEM-OP]_RAMFILE,                ;Store in AC
         GOTOP [IFETCH]
```

```
.TOC    "       Boolean Instructions"

;Here for SETZx
SETZX:  ZERO_[MEM-OP],
         DISPATCH [INST EXCT]

;Here for ANDx
ANDX:    [AC-OP]&[MEM-OP]_B,             ;And AC with MEM and store
         DISPATCH [OPERAND STORE]

;Here for ANDCAx
ANDCAX: [AC-OP]BAR&[MEM-OP]_B,           ;And AC complement with mem and store
         DISPATCH [OPERAND STORE]

;Here for SETMI = XMOVEI
XMOVEI: CALL [E_MEM-OP]
        SEL AC+[0],
         DISPATCH [OPERAND STORE]

;Here for ANDCMx
ANDCMX: COMPLEMENT [MEM-OP]_F,           ;Complement mem operand then
         F_[MEM-OP],GOTO [ANDX]          ; same as AND

;Here for XORx
XORX:    [AC-OP].XOR.[MEM-OP]_F,F_B,     ;Xor operands and store
         DISPATCH [OPERAND STORE]

;Here for IORx
IORX:    [AC-OP].OR.[MEM-OP]_B,          ;Ior operands and store
         DISPATCH [OPERAND STORE]
```

```
;Here for ANDCBx
ANDCBX:
;Small way is:
;       COMPLEMENT [AC-OP]_F,           ;Complement AC operand
;        F_[AC-OP],
;        GOTO [ANDCMX]
;Fast way is:
        COMPLEMENT [MEM-OP]_F,          ;Complement mem operand
         F_[MEM-OP]
        [AC-OP]BAR&[MEM-OP]_B,          ;And complements and store
         DISPATCH [OPERAND STORE]

;Here for EQVx
EQVX:   [AC-OP].XNOR.[MEM-OP]_F,F_B,    ;Put eqv (XNOR) in mem store
         DISPATCH [OPERAND STORE]

;Here for SETCAx
SETCAX: COMPLEMENT [MEM-OP]_F,          ;Complement AC and store
         F_[MEM-OP],
         DISPATCH [OPERAND STORE]

;Here for ORCAx
ORCAX:  COMPLEMENT [AC-OP]_F,           ;Complement AC then same as OR
         F_[AC-OP],
         GOTO [IORX]

;Here for SETCMx
SETCMX: COMPLEMENT [MEM-OP]_F,          ;Complement mem and store
         F_[MEM-OP],
         DISPATCH [OPERAND STORE]

;Here for ORCMx
ORCMX:  COMPLEMENT [MEM-OP]_F,          ;Complement mem then same as OR
         F_[MEM-OP],
         GOTO [IORX]

;Here for ORCBx
ORCBX:  [AC-OP]&[MEM-OP]_B,             ;And operands then same as SETCM
         GOTO [SETCMX]

;Here for SETOx
SETOX:  ONES_[MEM-OP],
         DISPATCH [INST EXCT]
```

```
.TOC    "       Half Word"

;Here for HLL
HLL:    [AC-OP]_[MEM-OP RH],            ;Preserve RH of AC then store
         DISPATCH [OPERAND STORE]

;Here for HLLI = XHLLI
XHLLI:  [E]_[MEM-OP LH],SECTION SELECT  ;Preserve RH of AC
        [K 7777.-1]&[MEM-OP]_B,         ;Strip extraneous bits
         IF [NOT AC REF] THEN [TO AC.0]
        [PC]_Y,SECTION SELECT,          ;Check PC section
         IF [GLOBAL] THEN [TO AC.0]
        SEL AC+[0],
         IF [SECTION 0] D [OPERAND STORE]
        [BIT17]_[MEM-OP LH],            ;Sect = 1 here
         GOTO [TO AC]

;Here for HRLM
HRLM:   SWAP [AC-OP]_[AC-OP]            ;Swap AC op then same as HLLM
        ; GOTO [HLLM]

;Here for HLLM
HLLM:   [AC-OP]_[MEM-OP LH],            ;Preserve LH of AC then store
         DISPATCH [OPERAND STORE]

;Here for HRL & HRLI
HRL:    SWAP [MEM-OP]_[AC-OP LH],       ;Swap memory op then same as HLL
         DISPATCH [OPERAND STORE]

;Here for HRLS
HRLS:   SWAP [MEM-OP]_[MEM-OP LH],      ;Put RH in LH then store
         DISPATCH [OPERAND STORE]

;Here for HLLZX
HLLZX:  [K -1.0]&[MEM-OP]_B,            ;Mask off rh and store
         DISPATCH [DISPATCH 4]

;Here for HRLEX
HRLEX:  [MEM-OP]_Y,IX_MX                ;Test sign bit
        ; GOTO [HLLEX]

;Here for HLLEX
HLLEX:  ZERO_[MEM-OP RH],               ;Clear RH
         IF NOT CT [MN] D [DISPATCH 4]
        ;GOTO [HLLOX]

;Here for HLLOX
HLLOX:  ONES_[MEM-OP RH],               ;Add ones to rh and store
         DISPATCH [DISPATCH 4]
```

```
;Here for HRR & HRRI
HRR:    [AC-OP]_[MEM-OP LH],            ;Preserve AC's lh then store
         DISPATCH [OPERAND STORE]

;Here for HLRM
HLRM:   SWAP [AC-OP]_[AC-OP]            ;Swap AC arg then same as HRRM
        ; GOTO [HRRM]

;Here for HRRM
HRRM:   [AC-OP]_[MEM-OP RH],            ;Copy Rh then store
         DISPATCH [OPERAND STORE]

;Here for HLR & HLRI
HLR:    SWAP [MEM-OP]_[AC-OP RH],       ;Swap mem arg then same as HRR
         DISPATCH [OPERAND STORE]

;Here for HLRS
HLRS:   SWAP [MEM-OP]_[MEM-OP RH],      ;Put LH in RH then store
         GOTO [TO SELF]

;Here for HRRZX
HRRZX:  ZERO_[MEM-OP LH],               ;Clear LH and store
         DISPATCH [DISPATCH 4]

;Here for HRREX
HRREX:  [K -1.0]BAR&[MEM-OP]_E,         ;Clear RH and check sign
         IF [NOT B BIT 18] D [DISPATCH 4]
        ;GOTO [HRROX]

;Here for HRROX
HRROX:  ONES_[MEM-OP LH],               ;Put ones in LH and store operand
         DISPATCH [DISPATCH 4]
```

```
.TOC    "       Logical Testing and Modification"

;Here for TRxx Instructions to get test bits and AC
TRX:    ZERO_[E LH],                    ;Leave only E RH, then set AC
         GOTO [TDXX]

;Here for TLxx Instructions to get test bits and AC
TLX:    ZERO_[E LH]                     ;Strip RH and select AC
        SWAP [E]_[E],                   ;Put test bits in LH and select AC
         GOTO [TDXX]

;Here for TDxx Instructions to get test bits and AC
TDX:    READ [E]                        ;Fetch test bits
        [MB]_[E],SEL AC+[0]             ;Copy test bits, and select AC
TDXX:   RAMFILE_[MEM-OP],NRFRD/1,       ;Get AC and dispatch
         SET LOCAL,
         DISPATCH [DISPATCH 2]

;Here for TSxx Instructions to get test bits and AC
TSX:    READ [E]                        ;Fetch test bits
        SWAP [MB]_[E],SEL AC+[0]        ;Swap test bits and select AC
        RAMFILE_[MEM-OP],NRFRD/1,       ;Get AC and dispatch
         SET LOCAL,
         DISPATCH [DISPATCH 2]

;Here for TxE Instructions for conditional skip
TXE:    [E]&[MEM-OP]_Y,                 ;Check for need to skip
         IF NOT CT [IZ] D [DISPATCH 3]
        ;GOTO [TXA]

;Here for TxA Instructions for conditional skip
TXA:    [PC]+1_[PC RH],
         DISPATCH [DISPATCH 3]

;Here for TxN Instructions for conditional skip
TXN:    [E]&[MEM-OP]_Y,                 ;Check for need to skip
         IF CT [IZ] D [DISPATCH 3]
        [PC]+1_[PC RH],
         DISPATCH [DISPATCH 3]

;Here for TxZ Instructions to change bits
TXZ:    [E]BAR&[MEM-OP]_Y,              ;Store new bits
         ALU_RAMFILE,
         GOTOP [IFETCH]

;Here for TxC Instructions to change bits
TXC:    [E].XOR.[MEM-OP]_Y,             ;Store new bits
         ALU_RAMFILE,
         GOTOP [IFETCH]

;Here for TxO Instructions to change bits
TXO:    [E].OR.[MEM-OP]_Y,              ;Store new bits
         ALU_RAMFILE,
         GOTOP [IFETCH]
```

```
.TOC    "EXTEND INSTRUCTIONS"

.IF/FTEXTEND
EXTEND: READ [E]                        ;Get EO word
        CALL [EFA CALC]
.ENDIF/FTEXTEND
```

```
.TOC    "CIS Instructions"

.IF/FTCIS
; Current block AC0 will be as follows for interrupts
;                   1111111111222222 222333333
;         01234567890123456789012345 6789012345
;        +--------------------------+---------+
;        +                          +         +
;        +--------------------------+---------+
;                                    \       /
;                                        + --- number of bytes so far
;
;Register usage
;        W1 current byte
;        W2 src adr (bit0=immediate;bits 4-35 are 32bit byte adr)
;        W3 src wrd
;        W4 dest adr
;        W5 dest wrd
;        W6 0,,bytes left to do
;        AC0 number of bytes to do,,number of bytes written

;Common routine to do setup for CIS instructions
CIS SETUP:
        J[400]_[W1]
        SWAP [W1]_[W1]                  ;Make a bit9
        [W1]_Q                          ;Save copy in Q reg
        Q&[IR]_F,F_Y LSRQ_Q,IX_MX       ;Test bit 9
        Q&[IR]_F,F_Y LSRQ_Q,            ;Test bit 10
         IF CT [MZ IX_MX] THEN [CS 2]
;Here if bit 9 is 0
        Q&[IR]_F,F_Y LSRQ_Q,            ;Test bit 11
         IF NOT CT [MZ IX_MX] THEN [CS 1]
        [E]_[W2]                        ;Copy source adr
        2*[W2]_[W2],
         IF NOT CT [IZ] CALL [W2+1]
        2*[W2]_[W2],
         IF NOT CT [IZ] CALL [W2+1]
        [PC FLAGS]&[BIT4]_F             ;Check first part done
W2+1:   [W2]+1_[W2],RETURN
;Here if bits 9,10 are 0,1
CS 1:   Q&[IR]_F,F_Y LSRQ_Q,            ;Test bit 12
         IF NOT CT [MZ IX_MX] THEN [MUUO]
        [E]_[W3],                       ;In case immediate
         IF NOT CT [MZ] THEN [CS1 IM]
;Here if first operand is deferred mode
        READ [E]                        ;Get adr of operand
.IF/FTMBZ
        J[32B MASK]_RAMFILE ADR
        RAMFILE_[W1],NRFRD/1
        [W1]BAR&[MB]_Y,IX_MX            ;Test for illegal bits
        [MB]_[W2],                      ;Copy adr
         IF NOT CT [MZ] THEN [MUUO]
.END!F/FTMBZ
.IFNOT/FTMBZ
        [MB]_[W2]                       ;Copy adr
.ENDIF/FTMBZ
;Here if immediate operand
CS1 IM:
;Here is bit 9 is 1
```

```
CS 2:   DISPATCH [DISPATCH 2]
```

```
MOVC:    [BIT4]&[PC FLAGS]_F            ;Check first part done
         JS[0]_[W6]
         LRS [W6]_[W6]
         LRS [W6]_[W6]
MOVC 7:  CALL [NXT SRC BYT]             ;Get next byte to transfer
         CALL [W NXT DST BYT]           ;Copy it
         [K -1]+[W6]_B,
          IF NOT CT [IZ] THEN [MOVC 7]
         GOTOP [IFETCH]
```

```
;Register conventions same as MOVC
CMPC:
CMPC 7: CALL [NXT SRC BYT]                ;Get first operand byte
        J[EIS W1]_RAMFILE ADR
        [K 777]&[W1]_Y,ALU_RAMFILE,       ;Save first operand
         CALL [R NXT DST BYT]             ; Get other byte
        [K 777]&[W1]_Q,
         J[EIS W1]_RAMFILE ADR
        RAMFILE_[W1],NRFRD/1
        Q-[W1]_Y,
         IF NOT CT [IZ IX_MX] THEN [CMPC 8]
        [K -1]+[W6]_B,
         IF NOT CT [IZ] THEN [CMPC 7]
        GOTOP [IFETCH]
CMPC 8: IF CT [MN] THEN [CMPC 9]
CMPC 9: GOTOP [IFETCH]
```

```
;Move Characters Variable Length
MOVCV:
;Compare Characters Variable Length
CMPCV:
;Compare Numeric Display
CMPND:
;Add Numeric Display
ADDND:
;Subtract Numeric Display
SUBND:
;Move Numeric Display
MOVND:
;Arithmetic Shift Numeric Display
ASHND:
;Convert Numeric Display to Binary
CVTNDB:
;Convert Binary to Numeric Display
CVTBND:
;Test for Legal Numeric Display
TLGND:
;Compare Packed
CMPP:
;Add Packed
ADDP:
;Subtract Packed
SUBP:
;Move Packed
MOVP:
;Arithmetic Shift Packed
ASHP:
;Clear High Order Packed
CHOP:
;Convert Packed to Binary
CVTPB:
;Convert Binary to Packed
CVTBP:
;Convert Numeric Display to Packed
CVTNDP:
;Convert Packed to Numeric Display
CVTPND:
;Convert EBCDIC Numeric Display to Packed
CVTEP:
;Convert Packed to EBCDIC Numeric Display
CVTPE:
```

```
;Here to read the next source byte
NXT SRC BYT:
        [W2]+1_[W2],                        ;Next byte adr
         IF CT [IN IX_MX] THEN [NXT SRC 1]
        ASR [W2]_[E] MC                     ;Copy adr to E
        ASR [W2]_[E] MC,                    ;Copy word adr
         IF CT [MC] THEN [NSB 2]            ; Check for odd or even byte
         IF CT [MC] THEN [NSB 3]
;Here if need to fetch next source word
        READ [E]
        [MB]_[W3],                          ;Copy data
         J[2]_CTR
        SWAP [W3]_[W1]
        2*[W1]_[W1],
         GOTO [NSB 30]

;Here for 3rd byte in word
NSB 3:  2*[W3]_[W1],                        ;Copy to right word
         J[3]_CTR
NSB 30: 4*[W1]_B,
         LOOP [NSB 30]
SWAP W1 W1:
        SWAP [W1]_[W1],
         RETURN

;Here for 2nd or 4th byte in word
NSB 2:  [W3]_[W1],
         IF CT [MC] RETURN
;Here for 2nd byte in word
        SWAP [W1]_[W1],
         RETURN

NXT SRC 1:
        [W3]_[W1],                          ;Next byte
         RETURN

;Here to read the next destination byte
R NXT DST BYT:
```

```
;Here to write the next destination byte
; Call with byte right justified in W1
 NXT DST BYT:
        [W4]+1_[W4]                     ;Next byte adr
        ASR [W4]_[E] MC                 ;Copy adr to E
        ASR [W4]_[E] MC,                ;Copy word adr
         IF CT [MC] THEN [WNDB 2]       ; Check for odd or even byte
         IF CT [MC] THEN [WNDB 3]
;Here if need to fetch next source word
        READ [E]
        [MB]_[W5],                      ;Copy data
         J[3]_CTR
        J[NBBYTE1]_RAMFILE ADR
        SWAP [W1]_[W1]
        2*[W1]_[W1],
         GOTO [WNDB 30]

;Here for 3rd byte in word
WNDB 3: J[NBBYTE3]_RAMFILE ADR
        2*[W5]_[W1],                    ;Copy to right word
         J[3]_CTR
WNDB 30:4*[W1]_B,
         LOOP [WNDB 30]
        RAMFILE_[MB],                   ;Get mask for character
         RETURN

WNDB 2: SWAP [K 777]_[MB],              ;Mask for byte
         IF CT [MC] THEN [WNDB 4]
;Here for 2nd byte in word
        SWAP [W1]_[W1]                  ;Position data
WNDB 20:[MB]BAR&[W5]_B                  ;Strip old byte from word
        [MB]&[W1]_B                     ;Strip data to move
        [W1].OR.[W5]_B,                 ;Add new bits to word
         RETURN

;Here for 4th byte in word
WNDB 4: [K 777]BAR&[W5]_Q               ;Mask off old data
        [K 777]&[W1]_B                  ;Mask off extraneous bits from data
        Q.OR.[W1]_[MB],                 ;Add new byte to data
         CALL [MEMORY WRITE]
        [E]+1_Q_[E],
         SPEC SEL/PAGE TABLE ENTRY,
         CALL [MEMORY READ]
        [MB]_[W5],
         RETURN
.ENDIF/FTCIS
```

```
.TOC    "IO INSTRUCTIONS"

IO:     2*[IR]_[W1],                    ;Copy IR to W1 shifted left
         IF [USER] CALL [CHECK IO OK]
        4*[W1]_B,                       ;Keep shifting
         CALL [2*W1_W1 LOAD IR]
        DISPATCH [IO 0]

2*W1_W1 LOAD IR:
        2*[W1]_[W1],                    ;Keep shifting
         LOAD IR,RETURN

IO 4:

;Here for an illegal IO instruction
XIO:    IF [USER] CALL [CHECK IO OK]
        GOTOP [IFETCH]

;Here to check if EXEC or user IO is set
; If IO is legal return, else do MUUO
CHECK IO OK:
        J[4000]_[W1],IF [EXEC] RETURN   ;Bit 24
        SWAP [W1]_[W1],IF [PXCT] RETURN ;Make bit 6=User I/O
        [W1]&[PC FLAGS]_Y,
         IF NOT CT [IZ] RETURN
        GOTOP [MUUO]
```

```
.TOC    "       Arithmetic Processor"

.NOBIN
Arithmetic Processor Identification
;       APRID
;       +-------+---+---+---+
;       ! 70000 ! ! ! X ! Y !
;       +-------+---+---+---+
;
; 0 1 2 3 4 5 6 7 8 9101112131415161718192021222324252627282930313233343 5
; ! \       / \                       / \                                 /
; !      !                !                              + --- reserved
; !      !                + --- u-code version number
; !      + --- reserved
; + --- u-code includes address break
.BIN

APRID:
;Possible coding
;       J[X]_[MEM-OP]                   ;Load bits 0-11
;       ROL [MEM-OP]_[MEM-OP],J[10.]_CTR
;       ROL [MEM-OP]_[MEM-OP],LOOP [.]
;       [MEM-OP]J[Y]_[MEM-OP]           ;Add bits 12-23
;       ROL [MEM-OP]_[MEM-OP],J[10.]_CTR
;       ROL [MEM-OP]_[MEM-OP],LOOP [.]
;       [MEM-OP]J[Z]_[MEM-OP],          ;Add bits 24-35
;        GOTO [TO MEMORY]
.IFNOT/FTADRB
        J[U-CODE VERSION]_[MEM-OP]      ;Put u-code version in register
.ENDIF/FTADRB
.IF/FTADRB
        J[U-CODE VERSION]_Y24-Y35,      ;Put u-code version in register
         [BIT17]_F,F_ROR Y_[MEM-OP]     ;Bit for address break feature
.ENDIF/FTADRB
        SWAP [MEM-OP]_[MEM-OP],         ;Put u-code version in lh
         GOTO [TO MEM]
```

```
.NOBIN
;Conditions Out, arithmetic processor
;       CONO APR,
;       +-------+---+---+---+
;       ! 70020 ! ! ! X ! Y !
;       +-------+---+---+---+
;
; E bits
;       18192021222324252627282930313233343 5
;        ! ! ! ! ! ! \    flags    / ! \   /
;        ! ! ! ! ! !                !   + --- PIA
;        ! ! ! ! ! !                + --- should be zero
;        ! ! ! ! ! + --- Set selected flags
;        ! ! ! ! + --- Clear selected flags
;        ! ! ! + --- disable selected flags
;        ! ! + --- enable selected flags
;        ! + --- clear all IO devices
;        + --- should be zero
.BIN

CONO APR:
.IFNOT/FT7PI
        J[6]_[W1]                       ;Mask for PI channel
        [W1]&[E]_Y,IX_MX
        ROR [W1]_[W1],                  ;Make mask for bit 35
         IF NOT CT [MZ] THEN [CNAPR1]
        [W1]&[E]_Y,                     ;May not put on channel 1
         IF NOT CT [IZ] THEN [MUUO]
CNAPR1:
.ENDIF/FT7PI
.IF/FTMBZ
        J[0010]_Y24-Y35,[BIT17]_F,      ;Makes 0,,400010
         F_ROR Y_[W1]
        [W1]&[E]_Y,                     ;Check for illegitimate bits
         IF NOT CT [IZ] THEN [MUUO]
.ENDIF/FTMBZ
        J[4000]_[W1]                    ;Mask to test bits
        [W1]+[W1]_Q,                    ;Put mask in Q reg
         J[APR FLAGS]_RAMFILE ADR
        J[7760]_[W2]                    ;Mask for flags
        [E]&[W2]_B                      ;Leave only flags
        RAMFILE_[W1],                   ;Get processor flags
         CALL [CAPR X]                  ; Turn on/turn off flags
        SWAP [W2]_[W2],                 ;Swap halves
         CALL [CAPR Y]                  ; Turn off/turn on flags
        J[7]_[W2]                       ;Mask for PI chnl
        [W2]BAR&[W1]_B                  ;Strip old PI chnl
        [E]&[W2]_B                      ;Leave only PI chnl
.IFNOT/FT7PI
        [W2]+[K -1]_Y,                  ;Check for setting PI level 1
         IF CT [IZ] THEN [MUUO]         ;No you don't
.ENDIF/FT7PI
        [W2].OR.[W1]_Y,ALU_RAMFILE      ;Save updated APR flags
        Q&[E]_F,ALU_Y/YES,2*Q_Q,        ;Test for IO reset
         IF NOT CT [IZ] CALL [RESET IO]
        CALL [SET PROC REG]
        SET LOCAL,
         GOTOP [IFETCH]
```

```
;Here to perform an IO reset function
; Wave IO RESET, clear all APR flags, clear all TTY flags
RESET IO:
        J[APR FLAGS]_RAMFILE ADR        ;Address APR flags
        ZERO_Y,ALU_RAMFILE              ;Clear flags
        J[LNOSW]_[W6],Y_RAMFILE ADR     ;Address first TTY link block
        J[4200]_[W5]                    ;CR5=RTS, CR1=DTR
        SWAP [W5]_[W5]
        4*[W5]_B,ALU_RAMFILE            ;Save initial CTY status
        [W6]+1_F,F_Q,J[75]_CTR          ;Address mode word
IOR2:   Q+1_F,F_Q_Y,Y_RAMFILE ADR       ;Address next location
        ZERO_Y,ALU_RAMFILE,LOOP [IOR2]
        ;J[2000]_[W1]                   ;Bit for IO reset
        ;[W1]_Y,ALU_PROC REG,           ;Tell processor reset
        ; CALL [SET PROC REG]           ; Then tell it right info
        ZERO_Y,ALU_PROC REG,            ;Clean processor reg
         CALL [SET PROC REG]            ; Then load it
        GOTO [SET PCI MODE]             ;Set mode for CTY
```

```
;Here to test for set/test for clear flags
; Mask (to be shifted left) is in Q and is tested with E
; W2 is ored or bar&ed into W1
CAPR X: Q&[E]_F,ALU_Y/YES,2*Q_Q,        ;Test for set selected flags/enables
         IF NOT CT [IZ] CALL [W2.OR.W1] ;Set selected flags/enables
        Q&[E]_F,ALU_Y/YES,2*Q_Q,        ;Test for clear selected flags/enables
         IF CT [IZ] RETURN
W2BAR&W1_B:
        [W2]BAR&[W1]_B,                 ;Clear selected flags/enables
         RETURN

;Here to test for clear/test for set flags
; Mask (to be shifted left) is in Q and is tested with E
; W2 is ored or bar&ed into W1
CAPR Y: Q&[E]_F,ALU_Y/YES,2*Q_Q,        ;Test for set selected flags/enables
         IF CT [IZ] THEN [CAPRY2]
        [W2]BAR&[W1]_B                  ;Clear selected flags/enables
CAPRY2: Q&[E]_F,ALU_Y/YES,2*Q_Q,        ;Test for clear selected flags/enables
         IF CT [IZ] RETURN
W2.OR.W1:
        [W2].OR.[W1]_B,                 ;Set selected flags/enables
         RETURN
```

```
.NOBIN
;Conditions In, arithmetic processor
;       CONI APR,
;       +-------+---+---+---+
;       ! 70024 ! I ! X ! Y !
;       +-------+---+---+---+
;
; 0 1 2 3 4 5 6 7 8 9101112131415161718192021222324252627282930313233343 5
;             \    flags    /                     \    flags     / ! \   /
;                   !                                   !        !   + --- PIA
;                   !                                   !        + --- intrupt req
;                   !                                   + --- flags
;                   + --- flags enabled for interrupts
.BIN

CONI APR:
        CALL [CI APR]
        GOTO [TO MEM]                   ;Store results and exit

;Here for CONI APR, CONSZ APR, or CONSO APR,.
CI APR: J[APR FLAGS]_RAMFILE ADR        ;Where we keep flags
        RAMFILE_[MEM-OP],NRFRD/1
        SWAP [MEM-OP]_[W1]              ;Put enables in RH
        [MEM-OP]&[W1]_Y,IX_MX           ;Mask bits
        J[0010]_[W1],                   ;Bit for interrupt request
         IF CT [MZ] RETURN
        [W1].OR.[MEM-OP]_B,
         RETURN

CONSZ APR:
        CALL [CI APR]                   ;Get APR bits for CONI
        ZERO_[MEM-OP LH],
         GOTO [CONSZ]

CONSO APR:
        CALL [CI APR]                   ;Get APR bits for CONI
        ZERO_[MEM-OP LH],
         GOTO [CONSO]
```

```
.TOC    "       PI System"
.NOBIN
;Conditions Out, Priority Interrupt
;       CONO PI,
;       +-------+---+---+---+
;       ! 70060 ! ! ! X ! Y !
;       +-------+---+---+---+
;
; E bits
;       18192021222324252627282930313233435
;        \     / ! ! ! ! ! ! ! 1 2 3 4 5 6 7
;         \   /  ! ! ! ! ! ! ! \           /
;          \ /   ! ! ! ! ! ! !       + --- Select levels
;           !    ! ! ! ! ! ! + --- Turn on PI system
;           !    ! ! ! ! ! + --- Turn off PI system
;           !    ! ! ! ! + --- Turn off level
;           !    ! ! ! + --- Turn on level
;           !    ! ! + --- initiate interrupt on level
;           !    ! + --- clear PI system
;           !    + --- drop program request on level
;           + --- should be zero
.BIN

CONO PI:
.IF/FT7PI
.IF/FTMBZ
        J[7400]_[W1]                    ;Mask for must be zero bits
        4*[W1]_B,                       ;Shift 6 places and check
         CALL [LLS4 MBZ]
.ENDIF/FTMBZ
        BIT0_[W2]                       ;Bit for PI on
        J[177]_[W3]                     ;Mask for chnl bits
        [W3]+1_Q,                       ;Put 200 in Q
         J[PI LVL REG]_RAMFILE ADR
        [E]&[W3]_B                      ;Leave only chnls to select
        RAMFILE_[W1],                   ;Get PI LVL REG from ramfile
         CALL [CAPR X]                  ; Test for PION/PIOFF
.ENDIF/FT7PI
.IFNOT/FT7PI
.IF/FTMBZ
        J[7401]_[W1]                    ;Mask for must be zero bits
        4*[W1]_B,                       ;Shift 6 places and check
         CALL [LLS4 MBZ]
.ENDIF/FTMBZ
        BIT0_[W2]                       ;Bit for PI on
        [K 77]_[W3]                     ;Mask for chnl bits
        [W3]+1_Q,                       ;Put 100 in Q
         J[PI LVL REG]_RAMFILE ADR
        [E]&[W3]_B                      ;Leave only chnls to select
        RAMFILE_Y,Y_[W1] QLS_Q,         ;Put PI LVL REG in W1
         QION_S100 0_Q100,              ; Put 200 in Q
         CALL [CAPR X]                  ; Test for PION/PIOFF
.ENDIF/FT7PI
        [W3]_[W2],                      ;Test for turn off/on levels
         CALL [CAPR Y]                  ; Set/clear bits in W1
        [W1]_RAMFILE                    ;Save updated PI LVL REG
        J[PI SFT REQ]_RAMFILE ADR
        Q&[E]_F,ALU_Y/YES,2*Q_Q,IX_MX   ;Test for initiate int on level
        RAMFILE_[W2],                   ;Get software interrupt requests
```

```
         IF CT [MZ] THEN [C PI 5]
        [W3].OR.[W2]_B,ALU_RAMFILE      ;Add new levels
C PI 5: Q&[E]_F,ALU_Y/YES,2*Q_Q,        ;Test for clear PI system
         IF CT [IZ] THEN [C PI 6]
        CALL [RESET PI]                 ;Reset the PI system
        SET LOCAL,GOTOP [IFETCH]

C PI 6: Q&[E]_Y,                        ;Test for drop int on level
         IF CT [IZ] THEN [C PI 7]
        [W3]BAR&[W2]_B,ALU_RAMFILE      ;Clear some levels
C PI 7: CALL [SET PI SYSTEM]            ;Setup PI hdw
        SET LOCAL,GOTOP [IFETCH]
```

```
;Here to reset the PI system
RESET PI:
        J[PI IN PROG]_RAMFILE ADR       ;Int's in progress in bits 29-35
        ZERO_Y,ALU_RAMFILE
        J[PI SFT REQ]_RAMFILE ADR       ;Software PI requests
        ZERO_Y,ALU_RAMFILE,
         INT OF/MASTER CLEAR            ; bits 29-35 are levels on
        J[PI LVL REG]_RAMFILE ADR       ;bit 0 is PI system on/off flag
        ZERO_[PI REG],ALU_RAMFILE,
         Y_2914 STATUS,
         GOTO [SET PI SYSTEM]           ;Set mask register

;Here to set up 2914 PI mask
SET PI MASK:
        J[PI LVL REG]_RAMFILE ADR       ;Address ramfile list of levels on
        J[300]_[W2]                     ;Always enable level 0 and 1
        RAMFILE_[W1]                    ;Get levels from ramfile
        [W1]_Y,
         IF NOT CT [IN] THEN [SPM 7]
        [W1].OR.[W2]_B                  ;Enable software levels also
SPM 7:  COMPLEMENT [W2]_F,F_Y,          ;Load mask with PI's to permit
         Y_2914 MASK,
         RETURN

;Here to setup PI system
SET PI SYSTEM:
        CALL [SET PI MASK]
        ;GOTO [SET PROC REG]
```

```
.NOBIN
;PROC REG (Processor register)
; 0 1 2 3 4 5 6 7 8 9101112131415161718192021222324252627282930313233343 5
;               \   /                                   \ / ! \   / \           /
;                 !                                     ! !   !         + --- PI req
;                 !                                     ! !   + --- Line PI reg
;                 !                                     ! + --- IO reset
;                 + --- Current AC block                + --- reserved
;
;Ramfile word [PROC REG]
; 0 1 2 3 4 5 6 7 8 9101112131415161718192021222324252627282930313233343 5
;               \   / \   /                             \ /   \   /
;                 !     + --- Prev ctxt AC block        !      + --- Line PI lvl
;                 + --- Current AC block                + --- reserved
.BIN

;Here to set up hardware PROC reg according to Ramfile
; Called whenever APR/TTY/SoftwarePIReq change
; Uses W1-W4
SET PROC REG:
        J[PI SFT REQ]_RAMFILE ADR       ;Ramfile list of software pi req's
        RAMFILE_[W4],NRFRD/1
        J[APR FLAGS]_RAMFILE ADR        ;Address APR flags
        RAMFILE_[W1],NRFRD/1            ;Get APR flags
        SWAP [W1]_[W2]                  ;Swap flags and enables
        [W1]&[W2]_Y,                    ;Check for flags with enables
         IF NOT CT [IZ] CALL [SPR 77]   :Set request level
        J[4000]_[W1]                    ;Will become 020000
        J[2220]_Y24-Y35,                ;Mask for flags = 022220
         [W1]+[W1]_F,F_LSZ Y_[W1]
        [W1]_Q,J[7]_CTR                 ;Save mask
        J[LNOSW]_[W3],Y_RAMFILE ADR     ;Address line block
SPR 2:  RAMFILE_[W1],NRFRD/1            ;Get status for line
        Q&[W1]_F,LSZ F_[W2],            ;Check for any flags set
         IF CT [IZ] THEN [SPR 4]
        [W1]&[W2]_Y,                    ;Check for enables for flags
         IF NOT CT [IZ] CALL [SPR 77]   ; Set int req
SPR 4:  J[10]_[W1]                      ;To make next line adr
        [W1]+[W3]_B,Y_RAMFILE ADR,      ;Next block adr
         LOOP [SPR 2]
        J[PROC REG]_RAMFILE ADR         ; Address ramfile copy of proc reg
        RAMFILE_[W3],NRFRD/1            ;Get ramfile copy of proc reg
SET RFPRG:
        J[RF PROC REG]_RAMFILE ADR
        [W4].OR.[W3]_Y,ALU_RAMFILE
        [W4].OR.[W3]_Y,ALU_PROC REG,    ;Change software interrupts
         RETURN

SPR 77: J[7]_[W2]                       ;Mask for PI level
        [W2]&[W1]_B,IF CT [IZ] RETURN   ;If no channel dismiss
        J[BIT28]_[W2]                   ;Base to index into bit table
        [W2]+[W1]_Y,Y_RAMFILE ADR       ;Make bit for our level
        RAMFILE_[W1],NRFRD/1            ;Get bit for interrupt
        [W1].OR.[W4]_B,RETURN           ;Set level
```

```
.NOBIN
;Conditions In, Priority Interrupt
;       CONI PI,
;       +-------+---+---+---+
;       ! 70064 ! ! ! X ! Y !
;       +-------+---+---+---+
;
; 0 1 2 3 4 5 6 7 8 9101112131415161718192021222324252627282930313233343 5
;                         1 2 3 4 5 6 7       1 2 3 4 5 6 7 ! 1 2 3 4 5 6 7
;                         \           /       \           / ! \           /
;                               !                   !       !       + --- levels on
;                               !                   !       + --- PI system on
;                               !                   + --- interrupt in progress on
;                               + --- program requests on
.BIN

CONI PI:
        CALL [CPI]                          ;Get PI bits
        J[PI SFT REQ]_RAMFILE ADR           ;Get software requests
        RAMFILE_[W1],NRFRD/1                ;Get software requests
        SWAP [W1]_[MEM-OP LH],
         GOTO [TO MEM]                      ;Give user result

;Here from CONI PI, or CONSO PI, or CONSZ PI
; Returns bits in MEM-OP, and 177 in W1
CPI:    J[PI LVL REG]_RAMFILE ADR           ;Get PI levels on
        J[177]_[W1]                         ;Mask for levels
        RAMFILE_Y,[W1]_F,F_Q Y_[W1]
        J[PI IN PROG]_RAMFILE ADR
        [W1]_Y,IX_MX                        ;Test for PI on
        Q&[W1]_[MEM-OP],                    ;Leave only Pi levels on
         IF NOT CT [MN] THEN [CPI2]         ; Branch if PIOFF
        Q+[MEM-OP]+1_F,F_[MEM-OP]           ;Add 200 to MEM-OP
CPI2:   RAMFILE_[W1],PUSH J[3]_CTR
        4*[W1]_B,RFCT                       ;Shift left 8 places
        [W1].OR.[MEM-OP]_B,                 ;Add interrupts in progress
         RETURN
```

```
;Conditions In and Skip if Zero, Priority Interrupt
CONSZ PI:
        CALL [CPI]                          ;Get bits to test in MEM-OP
CONSZ:  [MEM-OP]&[E]_Y,
         SET LOCAL,
         IF NOT CT [IZ] THENP [IFETCH]
        [PC]+1_[PC RH],
         GOTOP [IFETCH]

;Conditions In and Skip if Ones, Priority Interrupt
CONSO PI:
        CALL [CPI]                          ;Get bits to test in MEM-OP
CONSO:  [MEM-OP]&[E]_Y,
         SET LOCAL,
         IF CT [IZ] THENP [IFETCH]
        [PC]+1_[PC RH],
         GOTOP [IFETCH]
```

```
.TOC    "       Pager"
.NOBIN

;Conditions Out, Pager
;       CONO PAG, or WREBR
;       +-------+---+---+---+
;       ! 70120 ! ! ! X ! Y !
;       +-------+---+---+---+
;
; E format
;       18192021222324252627282930313233343 5
;        \   / ! ! \                        /
;         \ /  ! !           + --- Executive base address (page number)
;          !   ! + --- Enable Pager
;          !   + --- Tops20 Paging
;          + --- should be zero
.BIN

CONO PAG:
.IF/FTMBZ
        J[7000]_[W1]
        4*[W1]_B,
         CALL [LLS4 MBZ]
.ENDIF/FTMBZ
        CALL [PAGE TABLE CLEAR]         ;Clear entire page table
        [E]_[W1],                       ;Get page number
         CALL [W1(22) LLS9]
        J[BIT22]_RAMFILE ADR            ;Mask for enable
        Q&[W2]_[EPT],SET TOPS20         ;Set new EPT
        [E]_Q RAMFILE_[W2],CLEAR PAGED  ;Get bit for enable
        [W2]&[E]_F,ROR F_B ROR Q_Q,
         IF NOT CT [IZ] CALL [SET PAGED LATCH]
.IF/FT1OPAG
.IF/FT2OPAG
        Q&[W2]_Y,
         IF NOT CT [IZ] THEN [TO NOWHERE]
.ENDIF/FT2OPAG
        CLEAR TOPS20,
         GOTO [TO NOWHERE]
.ENDIF/FT1OPAG
.IFNOT/FT1OPAG
        SET LOCAL,
         GOTO [IFETCH]
.ENDIF/FT1OPAG

SET PAGED LATCH:
        SET PAGED,RETURN

;Here to shift W1 left 9 places and get 22 bit mask
W1(22) LLS9:
        J[22B MASK]_RAMFILE ADR         ;Mask for EPT adr

;Here to shift W1 left 9 places, leave it in Q and W1 and read ramfile into W2
W1 LLS9:
        2*[W1]_[W1],PUSH J[3]_CTR       ;Shift left 1 place
        4*[W1]_B,RFCT                   ;Keep shifting 2 at a time
        [W1]_Q RAMFILE_[W2],            ;Get ramfile
         RETURN
```

```
.IF/FTMBZ
LLS4 MBZ:
        4*[W1]_B
        4*[W1]_B
        [W1]&[E]_Y,
         IF CT [IZ] RETURN
        GOTO [MUUO]
.ENDIF/FTMBZ
```

```
.NOBIN

;Conditions In, Pager
;       CONI PAG, or RDEBR
;       +-------+---+---+---+
;       ! 70124 ! ! ! X ! Y !
;       +-------+---+---+---+
;
; E format
;       18192021222324252627282930313233343 5
;        \   / ! ! \                        /
;         \ /  ! !           + --- Executive base address (page number)
;          !   ! + --- Enable Pager
;          !   + --- Tops20 Paging
;          + --- reserved
.BIN

CONI PAG:
        J[BIT22]_RAMFILE ADR            ;Bit for paging enabled
        [EPT]_[W1],
         CALL [W1 LLS9]
        SWAP [W1]_[W1],                 ;Put page number in right half
         IF [PAGED] CALL [W2.OR.W1]     ;We are paged
        2*[W2]_[W2],                    ;Make a bit 21
         IF [TOPS20] CALL [W2.OR.W1]    ;Flag this is TOPS20
        [W1]_[MEM-OP],
         GOTO [TO MEM]
```

```
.NOBIN

)Data Out, Pager
;       DATAO PAG, or WRUBR
;       +-------+---+---+---+
;       ! 70114 ! ! ! X ! Y !
;       +-------+---+---+---+
;
; 0 1 2 3 4 5 6 7 8 9101112131415161718192021222324252627282930313233343 5
; ! ! !       \   / \   /   \       / !         \                       /
; ! ! !        \ /   \ /     \     /  !                    + --- User Base adr (page #)
; ! ! !         !     !       \ /    + --- Do not update accounts
; ! ! !         !     !        + --- previous context section
; ! ! !         !     + --- previous context ac block
; ! ! !         + --- current ac block
; ! ! + --- load user base address
; ! + --- select previous context section
; + --- select ac blocks
.BIN

DATAO PAG:
        READ [E]                        ;Get data word
        CALL [PAGE TABLE CLEAR]         ;Clear entire page table
        BITO_[W3]                       ;Get mask for enable
        J[PROC REG]_RAMFILE ADR,        ;Where we keep AC block numbers
         [W3]_Q                         ; Save mask in Q reg
        Q&[MB]_F,F_Y LSRQ_Q,IX_MX       ;Latch request to set AC blocks
        RAMFILE_[W3]                    ;Get old value for AC block numbers
        J[7700]_[W4]                    ;Build mask for AC blocks
        SWAP [W4]_[W4],
         IF CT [MZ] THEN [DATOP2]       ;If don't want to set blocks
        [W4]BAR&[W3]_B                  ;Strip old AC blocks
        [MB]&[W4]_B                     ;New AC blocks
        [W4].OR.[W3]_B,ALU_RAMFILE,
         CALL [SET RFPRG]
DATOP2: Q&[MB]_F,F_Y LSRQ_Q,            ;Check for select prev context sect
         IF CT [IZ] THEN [DATOP4]
        J[PCS]_RAMFILE ADR              ;Previous context section
        J[37]_[W1]                      ;Get mask for section
        SWAP [W1]_[W1]
        [W1]&[MB]_Y,ALU_RAMFILE         ;Save previous context section
DATOP4: Q&[MB]_Y,                       ;Check for load UBR
         IF CT [IZ] THENP [IFETCH]
        [MB]_[W1],                      ;Get page number
         CALL [W1(22) LLS9]
        Q&[W2]_[UPT],                   ;Set new UPT
         GOTOP [IFETCH]
```

```
.NOBIN

;Data In, Pager
;       DATAI PAG, or RDUBR
;       +-------+---+---+---+
;       ! 70104 ! I ! X ! Y !
;       +-------+---+---+---+
;
; 0 1 2 3 4 5 6 7 8 9101112131415161718192021222324252627282930313233343 5
; ! ! !       \   / \   /   \       /           \                         /
; ! ! !        \ /   \ /      \ /                          + --- User Base adr (page #)
; ! ! !         !     !        + --- previous context section
; ! ! !         !     + --- previous context ac block
; ! ! !         + --- current ac block
; ! ! + --- 1
; ! + --- 1
; + --- 1
.BIN

DATAI PAG:
        CALL [GET CONTEXT WD]           ;Get process context word
        [MB]_[MEM-OP],                  ;Put in right register
         GOTO [TO MEM]

;Here to get a "process context word"
GET CONTEXT WD:
        [UPT]_[W1],                     ;Put UPT page number in LH
         CALL [W1 LLS9]
        J[PCS]_RAMFILE ADR              ;Previous context section
        SWAP [W1]_[W1]                  ;Put UPT page number in RH
        [W1]_Q RAMFILE_[W1]
        J[PROC REG]_RAMFILE ADR         ;Get current AC block number
        Q.OR.[W1]_[MB]
        RAMFILE_[W1]
        [W1].OR.[MB LH]_B,              ;Add AC blocks to word
         RETURN
```

```
;Write the SPT base address
IF/FT20PAG
WRSPB:  READ [E]                        ;Get data
        [MB]_[SPT],                     ;Set SPT base adr, no checking
         GOTOP [IFETCH]

;Read SPT base address
RDSPB:  [SPT]_[MEM-OP],                 ;Get information he wants
         GOTO [TO MEM]
```

```
;Write CSTmsk register

WRCSTM: READ [E]                        ;Get data
        J[CSTMSK]_RAMFILE ADR           ;Address ramfile
        [MB]_RAMFILE,                   ;Set mask register, no checking
         SET LOCAL,
         GOTOP [IFETCH]

;Read CSTmsk register
RDCSTM: J[CSTMSK]_RAMFILE ADR           ;Address ramfile location
        RAMFILE_[MEM-OP],NRFRD/1,       ;Get information he wants
         GOTO [TO MEM]


;Write process use register (CSTDATA)
WRPUR:  READ [E]                        ;Get data
        J[CSTDATA]_RAMFILE ADR          ;Address ramfile location for data
        [MB]_RAMFILE,                   ;Set data word (age) register
         SET LOCAL,
         GOTOP [IFETCH]

;Read process use register (CSTDATA)
RDPUR:  J[CSTDATA]_RAMFILE ADR          ;Address ramfile location for data
        RAMFILE_[MEM-OP],NRFRD/1,       ;Get information he wants
         GOTO [TO MEM]
```

```
;Write Core status table base address
WRCSB:  READ [E]                        ;Get data
        J[CST]_RAMFILE ADR              ;Ramfile adr of CST
        [MB]_RAMFILE,                   ;Set CST register, no checking
         SET LOCAL,
         GOTOP [IFETCH]

;Read Core status table base address
RDCSB:  J[CST]_RAMFILE ADR              ;Address Ramfile
        RAMFILE_[MEM-OP],NRFRD/1,       ;Get information he wants
         GOTO [TO MEM]
.ENDIF/FT2OPAG
```

```
CLRPT:  SET USER,CALL [CLR PTE]          ;Clear user entry first
        CLEAR USER,CALL [CLR PTE]        ;Now clear exec entry
        GOTOP [IFETCH]

CLR PTE:
        [E]_Y,SECTION SELECT,
         SPEC SEL/PAGE TABLE ENTRY
ZERO_RAMFILE:
        ZERO_Y,ALU_RAMFILE,              ;Clear entry
         RETURN
```

```
.TOC    "       Time"

;TIMER FLAG goes true 27 times/millisecond
;       note: 27.*37.=999.      note: 37.=45
;       note: 27.*148.=3996.    note: 148.=224
; Ramfile locations TIME & TIME+1 are 72bit u-second counter

RDTIM:  J[27.]_[MEM-OP]
        [MEM-OP]-[TIME]_A               ;Number of clock ticks this ms
        ZERO_[MEM-OP+1],CALL [RDT 4]
        CALL [RDT 4]
        J[TIME BASE+1]_RAMFILE ADR
        2*[MEM-OP]_[MEM-OP],
         CALL [RDT 4]
        RAMFILE_[MEM-OP]                ;Get low order part of uptime
        J[TIME BASE]_RAMFILE ADR
        [MEM-OP]+[MEM-OP+1]_B,IX_MX
        RAMFILE_[MEM-OP],
         IF NOT CT [MOVR] THEN [D TO MEM]
        [MEM-OP]+1_[MEM-OP],
         GOTO [D TO MEM]

;Shift MEM-OP left 2 then add to MEM-OP+1
RDT 4:  4*[MEM-OP]_B                    ;Shift two left
        [MEM-OP]+[MEM-OP+1]_B,
         RETURN

;Here for WRTIM instruction
;       Write time base register
WRTIM:  READ [E]                        ;Get high order bits
        J[TIME BASE]_RAMFILE ADR
        [MB]_Y,ALU_RAMFILE,             ;Write high order bits
         CALL [READ NEXT]
        J[TIME BASE+1]_RAMFILE ADR
        [MB]_RAMFILE,
         SET LOCAL,
         GOTOP [IFETCH]
```

```
;Here for RDINT instruction - read time interval register
RDINT:  J[TIME INTERVAL]_RAMFILE ADR     ;Address ramfile location he wants
        RAMFILE_[MEM-OP],NRFRD/1
        4*[MEM-OP]_B,                    ;Display as u-seconds
         GOTO [TO MEM]

;Here for WRINT instruction - write interval register
WRINT:  READ [E]
        J[INTERVAL COUNTER]_RAMFILE ADR ;Start interval counter again
        [MB]+1_F,B SEL/MB,
         LRS F_B,O_SION O_QION
        [MB]+1_F,B SEL/MB,
         LRS F_B,O_SION O_QION,
         ALU_RAMFILE
        J[TIME INTERVAL]_RAMFILE ADR     ;Save for restarting counter
        [MB]_RAMFILE,
         SET LOCAL,
         GOTOP [IFETCH]
```

```
.TOC    "       Halt Status Block"

WRHSB:  READ [E]                        ;Get new value for HSB
        J[HSB]_RAMFILE ADR              ;Address ramfile
        [MB]_RAMFILE,                   ;Put new value in ramfile
         SET LOCAL,
         GOTOP [IFETCH]

RDHSB:  J[HSB]_RAMFILE ADR              ;Address of halt status block
        RAMFILE_[MEM-OP],NRFRD/1,
         GOTO [TO MEM]
```

```
.TOC    "       PCI"

.NOBIN
;Status word in ramfile
; 0 1 2 3 4 5 6 7 8 9101112131415161718192021222324252627282930313233343 5
; ! ! \      CR     / ! !       \   / ! ! !         ! ! ! ! \ RCV / \   /
; ! !         !       ! !         !   ! ! !         ! ! ! ! ! ! ! !   + --- Pi chnl
; ! !         !       ! !         !   ! ! !         ! ! ! ! ! ! ! + --- xof flag
; ! !         !       ! !         !   ! ! !         ! ! ! ! ! ! + --- xof enable
; ! !         !       ! !         !   ! ! !         ! ! ! ! ! + --- rcv done
; ! !         !       ! !         !   ! ! !         ! ! ! ! + --- rcv int enable
; ! !         !       ! !         !   ! ! !         ! ! ! + --- xmt done
; ! !         !       ! !         !   ! ! !         ! ! + --- xmt enable
; ! !         !       ! !         !   ! ! !         ! + --- dataset change flag
; ! !         !       ! + --- SR6 !   ! ! !         + --- data set change enable
; ! !         !       + --- SR7   !   ! ! + --- Nxm or mem err on rcv
; ! !         + --- CR7...CR0     !   ! + --- Nxm or mem err on xmt
; ! + --- Half duplex.            !   + --- flag for TTYRCV BITS
; + --- DDT mode.                + --- protocol

;Mode word in ramfile
; 0 1 2 3 4 5 6 7 8 9101112131415161718192021222324252627282930313233343 5
; \     / \         / \           /     \                           /
;    !         !            + --- MR17-MR10            + --- CRC seed
;    !         + --- MR25-MR20
;    + --- reserved

;Syn word in ramfile
; 0 1 2 3 4 5 6 7 8 9101112131415161718192021222324252627282930313233343 5
;                          \            / \            / \            /
;                                !              !              + --- Syn1 char
;                               + --- DLE char  + --- Syn2 char

;CRC word in ramfile
; 0 1 2 3 4 5 6 7 8 9101112131415161718192021222324252627282930313233343 5
;     \                         /     \                          /
;                   + --- Incoming CRC                + --- Outgoing CRC

;Transmit and receive header words
; 0 1 2 3 4 5 6 7 8 9101112131415161718192021222324252627282930313233343 5
; \                       / \                                      /
;              + --- reserved                    + --- phys adr of current header

;Transmit and receive byte pointer words
; 0 1 2 3 4 5 6 7 8 9101112131415161718192021222324252627282930313233343 5
; \                     / \                                      / \ /
;           !                                 !                      + --- byte number
;           + --- byte count                  + --- phys adr of byte word
.BIN
```

```
.IF/DEBUGTTY
RDTTY:
        [E]_Y,Y_TTY ADR                 ;Address register
        PCI_ALU,Y_[MEM-OP]              ;Read register
        PCI_ALU,Y_[WI]                  ;Get 2nd half
        SWAP [WI]_[MEM-OP LH],
         GOTO [TO AC.O]
.ENDIF/DEBUGTTY
```

```
;Here to reset a PCI line
.IF/FTTTYR
RESET PCI:
        [K -1.0]&[W6]_Y,IX_MX           ;Check for CTY
        J[200]_[W5]                     ;Transmit done
        [W6]_Y,Y_RAMFILE ADR,           ;Address status word
         IF NOT CT [MZ] THEN [RPCI1]
        J[4200]_[W1]                    ;For RTS & DTR
        4*[W1]_B
        SWAP [W1]_[W5 LH]               ;Put in LH
RPCI1:  [W5]_Y,ALU_RAMFILE,             ;Update status word
         CALL [SET PCI CR]
        [W6]+1_F,F_Q_Y,Y_RAMFILE ADR,   ;Address of Mode word
         J[5]_CTR
        ZERO_[W1 RH]                    ;Clear CRC seed
        RAMFILE_[W1 LH],                ;Get mode word
         CALL [W1_RAMFILE]
RPCI2:  Q+1_F,F_Y,Y_RAMFILE ADR         ;Address next ramfile word
        ZERO_Y,ALU_RAMFILE,             ;Zero next location
         LOOP [RPCI2]
        ;GOTO [SET USART MODE]
        RETURN
.ENDIF/FTTTYR
```

```
;Here to set mode register in USART line
; Call with [W6]=line number,,adr of ramfile line block
SET PCI MODE:
        J[A1!A0]_[W1]                   ;Address bits for CR
        ZERO_[W2],CALL [WR PCI]         ;Clear command register
        [W6]+1_Y,Y_RAMFILE ADR,         ;Address mode word
         CALL [RAMFILE_W1]              ;Get mode word from ramfile
        SWAP [W1]_[W1]                  ;Put mode words in RH
        J[A1]_[W2]                      ;Makes A1 for mode register
        SWAP [W6]_[W3],                 ;Get line number
         CALL [SUM 40]                  ; write MR1 & MR2
        [W1]_Y,ALU_MR2                  ;Write M8626 copy of MR2
        J[A0]_[W2]                      ;Set A0 for SYN1/SYN2/DLE
        J[LNXSYN]_[W1]                  ;Offset for syn word
        [W6]+[W1]_Y,Y_RAMFILE ADR       ;Address ramfile word
        RAMFILE_[W1],NRFRD/1,           ;Get synchs and DLE chars
         CALL [SUM 40]                  ;Write SYN1, SYN2 registers
        CALL [SUM 1]                    ;Write Dle register
        [W6]_Y,Y_RAMFILE ADR            ;Address ramfile copy of CR
        RAMFILE_[W5],NRFRD/1,           ;Set CR according to ramfile
         GOTO [SET PCI CR]

;Here to set 2651 mode or syn registers
; Call with MR2, MR1 as low order 8bit bytes in W1,
;       and with A1 or A0 in W2 and line number in W3
SUM 40: [W2]+[W3]_Y,Y_TTY ADR           ;Address line and mode registers
        [W1]_Y,ALU_PCI                  ;Write MR1/SYN1
SUM 1:  4*[W1]_B,                       ;Shift left 2 at a time
         J[3]_CTR
SUM 2:  4*[W1]_B,                       ;Shift left 2 at a time
         LOOP [SUM 2]                   ; 10 all together
        SWAP [W1]_[W1]
        [W1]_Y,ALU_PCI,                 ;Write MR2/SYN2
         RETURN

;Here to set PCI CR
SET PCI CR:
;       SWAP [W5]_[W2],J[4.]_CTR        ;Copy CR
;SPCR2: 4*[W2]_B,LOOP [SPCR2]           ;Shift left 10 places
;       SWAP [W2]_[W2]                  ;Right justify CR
        ROL [W5]_[W2],J[8.]_CTR         ;Copy CR from status word
SPCR2:  ROL [W2]_[W2],LOOP [SPCR2]      ;Put CR in W2 rh
        [K -1.0]&[W6]_Y,IX_MX           ;Check for CTY line
        J[A1!A0]_[W1]                   ;Address bits for CR
        SWAP [W6]_[W1] F_Q,B[W1]_F      ;Get line number
        Q+[W1]_Y,Y_TTY ADR,             ;Address command register
         IF CT [MZ] THEN [SPCR7]
        [W2]_Y,ALU_PCI,                 ;Write CR
         RETURN
;Here when writing CR for CTY
SPCR7:  J[4]_[W1]                       ;Bits we always leave on in CTY
        [W1].OR.[W2]_Y,ALU_PCI,         ;Write CR
         RETURN
```

```
;Here for a transmitter interrupt for a PCI line
; call with adr of line status register in W6, and status in W5
INT XMT:
        [K -1.0]&[W6]_Y,                ;See if this is the cty
         IF NOT CT [IZ] THEN [XINT10]   ; Branch if not cty
        J[CMDFLG]_RAMFILE ADR           ;Address console flags
        RAMFILE_[W1],NRFRD/1
        J[CMRTKR]_RAMFILE ADR           ;Get taker for response buffer
.IF/FTCRAM
        RAMFILE_[W2],NRFRD/1            ;If no data stop transmitter
        [W2]_[W2],                      ;Let latches go
         IF CT [IZ] THEN [XINT05]
.ENDIF/FTCRAM
.IFNOT/FTCRAM
        RAMFILE_[W2],NRFRD/1,           ;If no data stop transmitter
         IF CT [IZ] THEN [XINT05]
.ENDIF/FTCRAM
        J[F.S]_[W2]                     ;Xoff flag
        [W1]&[W2]_Y,                    ;If Xoff stop typing
         IF NOT CT [IZ] THEN [XINT95]
        CALL [CMB NXT]                  ;Get next byte
        CALL [XMT CHR]                  ;Type it
        J[CMRPTR]_RAMFILE ADR           ;Get response putter
        RAMFILE_[W1],NRFRD/1
        J[CMRTKR]_RAMFILE ADR           ;Address response taker
        RAMFILE_[W2],NRFRD/1
        [W1].XOR.[W2]_Y,                        ;Check for buffer empty
         IF CT [IZ] CALL [CMRBUF FLUSH]
        GOTO [CMDC X]                   ;Process waiting commands if any

XINT05: [BIT4]&[W1]_Y,                  ;(F.USR) Exec mode=shut off xmtr
         IF CT [IZ] THEN [XINT95]

;Here because line is in user mode
XINT10:
        J[LNXXHD]_[W4]                  ;Offset for header
.IF/FTDDTM
        [W5]_Y,
         IF NOT CT [IN] THEN [XINT30]   ;Check for DDT mode

;Here because line is in DDT mode
        [W6]+[W4]+1_Y,Y_RAMFILE ADR     ;Address ramfile word
.IF/FTCRAM
        RAMFILE_[W1],NRFRD/1            ;Get next char to type
        [W1]_[W1],                      ;Let latches go
         IF CT [IZ] THEN [XINT95]       ; If nothing shut down
.ENDIF/FTCRAM
.IFNOT/FTCRAM
        RAMFILE_[W1],NRFRD/1,           ;Get next char to type
         IF CT [IZ] THEN [XINT95]       ; If nothing shut down
.ENDIF/FTCRAM
        ZERO_Y,ALU_RAMFILE,             ;Have sent char now
         CALL [XMT CHR]                 ; Type char
        CALL [SET XDN]                  ;Set done flag in status word
        GOTOP [PCI INT CLR]
.ENDIF/FTDDTM
```

```
;Here because line is not in DDT mode
XINT30: J[400]_[W1]                     ;Will become CR0
        SWAP [W1]_[W1]                  ;Makes CR0
        [W5]&[W1]_Y,
         IF CT [IZ] THEN [XINT95]
;Here to get next byte from buffer
        J[XMT TTY PF]_[W1]
        CALL [SET PF RCOVR]
        J[24B MASK]_RAMFILE ADR
        RAMFILE_[W2],NRFRD/1,           ;Get 24 bit mask
         CALL [NXT RX WRD]
        ROL [PMA]_[W1],IX_MX            ; byte #ab, a_MN
        ROL [W1]_[W1],                  ;Byte #ab, B_MN
         IF NOT CT [MN IX_MX] THEN [XINT70]
        IF NOT CT [MN] THEN [XINT72]    ;Check for byte 2 or 3
        GOTO [XINT73]
XINT70: SWAP [MB]_[MB],                 ;Enter here for byte 0
         IF CT [MN] THEN [XINT73]       ;Check for byte 0 or 1
XINT72: 2*[MB]_[MB],J[3]_CTR            ;Enter here for byte 2
XINT74: 4*[MB]_B,LOOP [XINT74]
        SWAP [MB]_[MB]                  ;Enter here for byte 1
XINT73: [MB]_[W1],CALL [XMT CHR]        ;Enter here for byte 3
        IF NOT CT [UZ] THENP [PCI INT CLR]      ;Check for buffer exhausted
XINT93: CALL [SET XDN]                  ;We are done with buffer
        J[LNXXHD]_[W4]                  ;Got clobbered so restore
        [W6]+[W4]_Y,Y_RAMFILE ADR       ;Addr RF copy of current hdr adr
        RAMFILE_[PMA],NRFRD/1,          ;Get adr of header
         CALL [CHNG2]                   ; write pointer back
        CALL [GET NXT RX HDR]           ;Get header for next buffer
        IF CT [UOVR] CALL [CLR XGO]     ;We have now stopped
PCI INT CLR:
.IF/DEBUG3
        J[POOP FLAG]_RAMFILE ADR
        ZERO_Y,ALU_RAMFILE
.ENDIF/DEBUG3
        ZERO_[W1],CALL [SET PF RCOVR]
        [PXCT]_[PI REG],Y_29[illegible]STATUS,  ;Restore previous level
         GOTOP [CONSOLE]                ;This isn't really an int

;Here because memory error reading transmit buffer
XINT91: ROR [BIT17]_[W1]                ;Need to make a bit 19
        ROR [W1]_[W1]                   ;Make a bit 19
        [W1].OR.[W5]_B,                 ;Set xmt err bit
         GOTO [XINT93]                  ; and stop transmitter

;Here when page fail (=nxm or mem fault) on xmt
XMT TTY PF:
        ROR [BIT17]_[W1]                ;Make a bit 18
        ROR [W1]_[W1]                   ;Make a bit 19
        [W1].OR.[W5]_B,                 ;Set xmt mem err flag
         CALL [SET XDN]                 ; And flag transmitter stopped
        CALL [CLR XGO]                  ;Clear CR0 in status word
;Here to stop the transmitter
XINT95: CALL [CLR PCI TXEN]             ;Disable transmitter
        GOTOP [PCI INT CLR]
```

```
;Here to get next word from transmit or receive buffer
; Call with LNXXHD or LNXRHD in W4 & 24b mask in W2
; Returns with word in MB
;       Returns UZ if count expires
NXT RX WRD:
        [W6]+[W4]+1_F,F_Y,Y_RAMFILE ADR ;Address byte pointer
        RAMFILE_[PMA],NRFRD/1           ;Get byte header
        [PMA]-[W2]_A,                   ;Increment pointer, decrement count
         ALU_RAMFILE                    ; save updated byte pointer
        [W2]BAR&[PMA]_Y,IX_UX           ;Set UZ if count has expired
        ROR [PMA]_[PMA]                 ;Address of word which has byte
        ROR [PMA]_[PMA],                ;Divide by 2
         GOTO [UNMAPPED READ 0]         ; get word
```

```
;Here to get next receive or transmit header
; CALL with LNXXHD or LNXRHD in W1
GET NXT RX HDR:
        [W6]+[W4]_Y,Y_RAMFILE ADR       ;Address ramfile
.IF/FTTTYSB
        RAMFILE_[PMA],NRFRD/1,          ;Get address of header
         CALL [UNMAPPED READ 0]         ;Get header status
        [BIT17]&[MB]_Y,                 ;Check for stop on buffer
         IF NOT CT [IZ] THEN [SET UOVR]
.ENDIF/FTTTYSB
.IFNOT/FTTTYSB
        RAMFILE_[PMA],NRFRD/1           ;Get adr of header
.ENDIF/FTTTYSB
        [PMA]+2_B,MEM START READ,       ;Makes adr of next header adr
         CALL [UNMAPPED READ]
        [W6]+[W4]_Y,Y_RAMFILE ADR       ;Address LNXXHD or LNXRHD
        [MB]_[PMA],ALU_RAMFILE,         ;Save adr of new header
         IF CT [IZ] THEN [SET UOVR]
GET RX HDR:
        [PMA]+1_[PMA],MEM START READ,   ;Get byte pointer for new header
         CALL [UNMAPPED READ]
        J[24B MASK]_RAMFILE ADR
        RAMFILE_[W1],NRFRD/1            ;Get mask for count
        [W1]BAR&[MB]_F,F_Y,             ;Check count field
         IF CT [IZ] THEN [SET UOVR]
        [W6]+[W4]+1_F,F_Y,Y_RAMFILE ADR
        [MB]_RAMFILE,0_UOVR,            ;Save new pointer
         RETURN
```

```
;Here to transmit a char on a 2651 line
; Call with [W6]=line number,,adr of ramfile line block, char in W1
XMT CHR:
        SWAP [W6]_[W2]                  ;Put line number in RH
        [W2]_Y,Y_TTY ADR                ;Address line and mode registers
        [W1]_Y,ALU_PCI,                 ;Transmit character
         RETURN

;Here to clear LS.RGO in LNXSWD
CLR RGO:
        J[2000]_[W1]                    ;Bit to clear
        [W6]_Y,Y_RAMFILE ADR,           ;Address the status word
         GOTO [CLR RXGO]

;Here to stop transmitter
CLR XGO:
        [W6]_Y,Y_RAMFILE ADR            ;Address the status word
        J[400]_[W1]                     ;Will become CRO
CLR RXGO:
        SWAP [W1]_[W1]                  ;Make CRO
        [W1]BAR&[W5]_B,ALU_RAMFILE,     ;Write updated status
         RETURN

;Here to set LS.RDN in LNXSWD
SET RDN:
        J[LS.RDN]_[W1]                  ;Bit to set
        [W6]_Y,Y_RAMFILE ADR,
         GOTO [W1!W5_RAMFILE]

.IF/FTTTYF
;Here to clear LS.RDN in LNXSWD
CLR RDN:
        J[LS.RDN]_[W1]                  ;Bit to set
        [W6]_Y,Y_RAMFILE ADR
W1BAR&W5_RAMFILE:
        [W1]BAR&[W5]_B,ALU_RAMFILE,     ;Write updated status
         GOTO [SET PROC REG]
.ENDIF/FTTTYF

;Here to set LS.XDN in LNXSWD
SET XDN:
        J[LS.XDN]_[W1]                  ;Bit to set
        [W6]_Y,Y_RAMFILE ADR
W1!W5_RAMFILE:
        [W1].OR.[W5]_B,ALU_RAMFILE,     ;Write updated status
         GOTO [SET PROC REG]            ; Adjust int requests

.IF/FTTTYF
;Here to clear LS.XDN in LNXSWD
CLR XDN:
        J[LS.XDN]_[W1]                  ;Bit to set
        [W6]_Y,Y_RAMFILE ADR
         GOTO [W1BAR&W5_RAMFILE]
.ENDIF/FTTTYF
```

```
;Here to set TXEN = transmit enable in a PCI
; call with     [W6] = line #,,x
;       clobbers W1 & W2
SET PCI TXEN:
        [BIT17]+1_[W2],                 ;Get bit for TXEN
         CALL [RD PCI CR]
        [W1].OR.[W2]_Y,ALU_PCI,         ;Set bit
         RETURN

;Here to clear TXEN = transmit enable in a PCI
; call with     [W6] = line #,,x
;       clobbers W1 & W2
CLR PCI TXEN:
        [BIT17]+1_[W2],                 ;Get bit for TXEN
         CALL [RD PCI CR]
        [W2]BAR&[W1]_Y,ALU_PCI,         ;Clear bit
         RETURN

;Here to set RXEN = receive enable in a PCI
; call with     [W6] = line #,,x
;       clobbers W1 & W2
SET PCI RXEN:
        J[RXEN]_[W2]                    ;Get bit for RXEN
        CALL [RD PCI CR]
        [W1].OR.[W2]_Y,ALU_PCI,         ;Set bit
         RETURN

;Here to clear RXEN = receive enable in a PCI
; call with     [W6] = line #,,x
;       clobbers W1 & W2
CLR PCI RXEN:
        J[RXEN]_[W2]                    ;Get bit for RXEN
        CALL [RD PCI CR]
        [W2]BAR&[W1]_Y,ALU_PCI,         ;Clear bit
         RETURN

;Here to read PCI SR register
RD PCI SR:
        J[A0]_[W1]                      ;Address of status register
        SWAP [W6]_[W1] F_Q,B[W1]_F,     ;Get line number
         GOTO [RDPCI2]

;Here to read PCI CR register
RD PCI CR:
        J[A1!A0]_[W1]                   ;Bits to address command register
        ;GOTO [RD PCI]

;Here to read a PCI register
RD PCI: SWAP [W6]_[W1] F_Q,B[W1]_F      ;Get line number
RDPCI2: Q+[W1]_Y,Y_TTY ADR              ;Address register
        PCI_ALU,Y_[W1],                 ;Read register
         RETURN

;Here to write a PCI register
; Call with Ax in W1, and data in W2
WR PCI: SWAP [W6]_[W1] F_Q,B[W1]_F      ;Get line number
        Q+[W1]_Y,Y_TTY ADR              ;Address register
        [W2]_Y,ALU_PCI.RETURN           ;Write register
```

```
;Here for a receiver interrupt for a PCI line
; call with adr of line status register in W6, and status in W1
INT RCV:
        J[LNXRHD]_[W4]                  ;Offset for receive header
        SWAP [W6]_[W1] F_Q,ZERO_F,      ;Get line number
         CALL [RDPCI2]                  ;Read receiver register
        J[377]_[W3]                     ;Mask to strip extraneous bits
        [W1]&[W3]_B,                    ;Strip extraneous bits
         CALL [RD PCI SR]               ; and get SR
        J[70]_[W2]                      ;Mask for error bits
        [W2]&[W1]_B,                    ;Leave only error flags
         IF CT [IZ IX_UX] THEN [RINT2]
        J[400]_[W2]                     ;Error flag
        [W2].OR.[W3]_B                  ;Add flag
        SWAP [W1]_[W3 LH]               ;Add flags
RINT2:  [K -1.0]&[W6]_Y,                ;See if this is the cty
         IF NOT CT [IZ] THEN [RINT10]   ; Branch if not cty
        J[177]_[W2]                     ;Mask to strip parity
        [W3]&[W2]_F,F_Q FJ[34]_[W2]     ;^\ char
        J[CMDFLG]_RAMFILE ADR           ;Address console flags
        Q-[W2]_Y,                       ;Check for typing ^\
         IF CT [IZ] THEN [CNTRL BKSLSH]
        RAMFILE_Y,Y_[W2]                ;Get console flags
        [BIT4]&[W2]_Y,                  ;(F.USR) Branch if user mode
         IF CT [IZ] THEN [CONSOLE CHAR] ;Treat as CTY input

;Here for a receive interrupt except for CTY in Console mode
; received char in W4
RINT10:
.IFNOT/FTDDTM
        [W4]+[W6]_Y,Y_RAMFILE ADR       ;Address char word
.ENDIF/FTDDTM
.IF/FTDDTM
        [W5]_Y,IX_MX                    ;Check status reg
        [W4]+[W6]_Y,Y_RAMFILE ADR,      ;Address char word
         IF NOT CT [MN] THEN [RINT30]   ; Branch if not DDT mode

;Here because line is in DDT mode
.IF/FTCRAM
        RAMFILE_[W2],NRFRD/1            ;Check for already holding
        [W2]_[W2],                      ;Let latches go
         IF CT [IZ] THEN [RINT12]
.ENDIF/FTCRAM
.IFNOT/FTCRAM
        RAMFILE_Y,NRFRD/1,              ;Check for already holding
         IF CT [IZ] THEN [RINT12]
.ENDIF/FTCRAM
        J[420]_[W2]                     ;Overrun error
        [W2].OR.[W1]_B                  ;Add error flag
RINT12: SWAP [W1]_[W4 LH]               ;Put flags in LH
        [W4]_RAMFILE                    ;Save character
        CALL [SET RDN]                  ;Set receiver done
        GOTOP [PCI INT CLR]
.ENDIF/FTDDTM
```

```
;Here because line is not in DDT mode
RINT30: J[2000]_[W1]                    ;Will become bit RXEN
        SWAP [W1]_[W1]                  ;Make RXEN
        [W5]&[W1]_Y,
         IF CT [IZ] THEN [RINT99]
;Here to put next byte into buffer
        J[RCV TTY PF]_[W1]
        CALL [SET PF RCOVR]
        J[24B MASK]_RAMFILE ADR
        RAMFILE_[W2],NRFRD/1,           ;Get 24 bit mask
         CALL [NXT RX WRD]              ; Get word for char
        J[0777]_[W1]                    ;Mask for data
        ROL [PMA]_[W2],IX_MX            ;Byte #ab, b_MN
        ROL [W2]_[W2],                  ;Byte #ab, a_MN
         IF NOT CT [MN IX_MX] THEN [RINT70]
        IF NOT CT [MN] THEN [RINT72]    ;Check for byte 2 or 3
        GOTO [RINT73]
RINT70: SWAP [W3]_[W3]                  ;Enter here for byte 0
        SWAP [W1]_[W1],                 ;Swap mask also
         IF CT [MN] THEN [RINT73]       ;Check for byte 0 or 1
RINT72: 2*[W3]_[W3]                     ;Enter here for byte 2
        2*[W1]_[W1],J[3]_CTR
RINT74: 4*[W3]_B
        4*[W1]_B,LOOP [RINT74]          ;Shift mask also
RINT73: [W1]BAR&[MB]_B                  ;Enter here for byte 3
                                        ; Clear slot for byte
        [W3].OR.[MB]_B,                 ;Add new byte
         CALL [UNMAPPED WRITE]          ; Save in buffer
        IF NOT CT [UZ] THEN [RINT99]    ;Check for buffer exhausted
RINT93: CALL [SET RDN]                  ;We are done with buffer
        J[LNXRHD]_[W4]                  ;Got clobbered so restore
        [W6]+[W4]_Y,Y_RAMFILE ADR       ;Addr RF copy of current hdr adr
        RAMFILE_[PMA],NRFRD/1,          ;Get adr of header
         CALL [CHNG2]                   ; write pointer back
        CALL [GET NXT RX HDR]   ;Get header for next buffer
        IF CT [UOVR] CALL [CLR RGO]     ;We have now stopped
RINT99: GOTOP [PCI INT CLR]             ;Dismiss interrupt

;Here because memory error on receive
RINT91: ROR [BIT17]_[W1],J[1]_CTR       ;Need to make a bit 20
RINT92: ROR [W1]_[W1],LOOP [RINT92]     ;Rotate to bit20
        [W1].OR.[W5]_B,                 ;Add rcv mem err flag
         GOTO [RINT93]                  ; and stop transmission

;Here when page fail (=nxm or mem fault) on rcv
RCV TTY PF:
        ROR [BIT17]_[W1],PUSH J[1]_CTR  ;Make a bit 18
        ROR [W1]_[W1],RFCT              ;Make a bit 20
        [W1].OR.[W5]_B,                 ;Set rcv mem err flag
         CALL [SET RDN]                 ; And flag transmitter stopped
        GOTO [PCI INT CLR]
```

```
;Here for a dataset change interrupt for a PCI line
INT DSC:
        CALL [RD PCI SR]                ; Get status register
        J[300]_[W2]                     ;Mask for DSD & DCD
        [W2]&[W1]_B,                    ;Leave only new DSD&DCD in W1
         CALL [SWAP W2_W2]              ;Put mask in LH
        [W2]&[W5]_Q,                    ;Put only DSD & DCD in Q
         CALL [SWAP W1_W1]              ;Put new status in LH
        Q.XOR.[W1]_F,F_Y,               ;Check for change
         IF CT [IZ] THEN [IDSC3]
        [W2]BAR&[W5]_B                  ;Clear old status bits
        J[4000]_[W2 RH]                 ;Get bit to make dataset change
        2*([W2]+[W2 RH])_B              ;Makes dataset change bit
        [W2].OR.[W5]_Y,                 ;Set bits
         ALU_RAMFILE
IDSC3:  GOTOP [PCI INT CLR]             ;Dismiss interrupt

SWAP W1_W1:
        SWAP [W1]_[W1],
         RETURN

SWAP W2_W2:
        SWAP [W2]_[W2],
         RETURN
```

```
;Here to accumulate CRC for character
; call with      [W1] char
;                [W2] CRC so far
;                [W3] CRC seed
.IF/FTCRC
CRC CALC:
        [W1].XOR.[W2]_F,F_B,
         J[7]_CTR
CRC 2:  [W2]_F,F LRS_[W2] S100_MC
        IF CT [MC] THEN [CRC 3]
        LOOP [CRC 2]
        RETURN
CRC 3:  [W3].XOR.[W2]_F,F_B,
         LOOP [CRC 2]
        RETURN
.ENDIF/FTCRC
```

```
.TOC    "       External"

IO EFA: IF [USER] CALL [CHECK IO OK]    ;Give users MUUO's
        DISPATCH [DISPATCH 2]
```

```
TIOE:   [AC-OP]&[MEM-OP]_Y,              ;Check for need to skip
         IF NOT CT [IZ] THENP [IFETCH],
         SET LOCAL
        [PC]+1_[PC RH],                  ;Skip
         GOTOP [IFETCH]

;TIOEB:


TION:   [AC-OP]&[MEM-OP]_Y,              ;Check for need to skip
         IF CT [IZ] THENP [IFETCH],
         SET LOCAL
        [PC]+1_[PC RH],                  ;Skip
         GOTOP [IFETCH]

;TIONB:
```

```
WRIO:   RAMFILE_[MEM-OP],NRFRD/1,
         GOTO [WRIO.1]

WRIO.1: CALL [WRIOX]
        SET LOCAL,
         IF NOT CT [UOVR] THENP [IFETCH]
        GOTOP [IO PF]                   ;Win an IO page fail

;Here to to an IO write
; Call with data in [MEM-OP] and adr in [E]
; returns with UOVR if illegal
WRIOX:  [K 7777.-1]&[E]_B,1_UOVR,       ;Strip extraneous bits
         CALL [VAL IO ADR]              ; Validate IO address
        [E]_[PMA],START IO WRITE,       ;Put out IO address
         IF CT [MN] THEN [WRIO TTY]
        [MEM-OP]_[MEM-OP],ALU_IO        ;Perform transfer
        0_UOVR,MEM HOLD,
         IF [NO BUS ERROR] RETURN
.IFNOT/FTCKBP
        IF [NOT MEM EXISTS] THEN [SET UOVR]
        IF [NOT MEM FAULT] RETURN
.ENDIF/FTCKBP
        1_UOVR,RETURN                   ;Set error flag

;Here to do an IO write to a TTY register
WRIO TTY:
        [E]-[W1]_F,F_Y,
         IF CT [IN IX_MX] RETURN
        RAMFILE_[W3],                   ;Get old contents of word
         IF CT [MZ] THEN [WRIOTM]
        [MEM-OP]_[W5],ALU_RAMFILE,      ;Write word
         0_UOVR
        [W2]&[E]_F,F_Q_Y,               ;Test which word was written
         IF CT [IZ] THEN [WRIOTO]       ; Branch for wrote status reg
.IF/FTTTYF
        J[LNXXCHR]_[W2]                 ;Offset for xmt char
        Q-[W2]_F,F_Y,IX_MX              ;See if wrote xmt char
        [W1]BAR&[W6]_B,                 ;Make adr of line block
         IF CT [IZ] THEN [WRIOT5]
.ENDIF/FTTTYF
        Q+[K -1]_F,F_Q_Y,
         IF CT [IZ] THEN [WRIOT12]
        Q+[K -1]_F,F_Q_Y,
         IF NOT CT [IZ] RETURN
WRIOT12:
        GOTO [SET PCI MODE]             ;Wrote Synch reg
```

```
;Here because wrote status register
WRIOTO:
.IFNOT/FT7PI
        J[6]_[W1]                       ;Mask for PI channel
        [W1]&[MEM-OP]_Y,IX_MX
        ROR [W1]_[W1],                  ;Make mask for bit 35
         IF NOT CT [MZ] THEN [WRIOT1]
        [W1]&[MEM-OP]_Y,                ;Check for putting on channel 1
         IF NOT CT [IZ] THEN [MUUO]     ; Putting on chnl 1 illegal
WRIOT1:
.ENDIF/FT7PI
.IF/FTTTYR
        J[4000]_[W1]                    ;Make a bit 23 from this
        ROL [W1]_[W1]                   ;Makes a bit 23
        [W1]&[MEM-OP]_Y,                ;Check for doing Reset
         IF NOT CT [IZ] THEN [RESET PCI]
.ENDIF/FTTTYR
        J[WRIO TMP]_RAMFILE ADR         ;Address temp location
        [W5].XOR.[W3]_F,F_B,ALU_RAMFILE ;Find flags which changed
        S SEL[W3],                      ;Check for transmit enable changed
         IF [B BIT 9] CALL [CHNG TXEN]
        J[BIT7]_RAMFILE ADR             ;Address a bit 7
        RAMFILE_[W1],NRFRD/1            ;Get a bit 7
        [W1]&[W3]_Y,                    ;Check for RXEN changed
         IF NOT CT [IZ] CALL [CHNG RXEN]
        CALL [SET PCI CR]               ;Set CR according to status word
        O_UOVR,                         ;Flag WRIO won
         GOTO [SET PROC REG]            ; Be sure ints are right

CHNG RXEN:
        J[LNXRHD]_[W4]                  ;Offset for receive header
        [W6]+[W4]_Y,Y_RAMFILE ADR,      ;Address ramfile copy
         1_UOVR                         ; of current hdr adr
        [W1]&[W5]_Y,                    ;Check for went off or on
         IF NOT CT [IZ] THEN [SET RXEN]
        GOTO [CHNG2]

CHNG TXEN:
        J[400]_[W1]
        SWAP [W1]_[W1]                  ;Make a bit 9 = CRO
        J[LNXXHD]_[W4]                  ;Offset for transmit header
        [W6]+[W4]_Y,Y_RAMFILE ADR,      ;Address ramfile copy
         1_UOVR                         ; of current hdr adr
        [W1]&[W5]_Y,                    ;Check went on or off
         IF NOT CT [IZ] THEN [SET TXEN]
CHNG2:  [W6]+[W4]+1_F,F_Y,Y_RAMFILE ADR ;Address ramfile copy of pointer
        RAMFILE_[MB],NRFRD/1,           ;Get ramfile copy of pointer
         CALL [ZERO_RAMFILE]            ;Clear ramfile copy
        [PMA]+1_[PMA],                  ;Make adr of pointer in header
         CALL [UNMAPPED WRITE]
        J[WRIO TMP]_RAMFILE ADR
        RAMFILE_[W3],NRFRD/1,           ;Get changed bits back
         RETURN

SET RXEN:
        RAMFILE_[PMA],                  ;Get current header adr
         IF NOT CT [IZ] CALL [GET RX HDR]
        IF CT [UOVR] CALL [CLR RGO]     ;In case lost
```

```
        GOTO [RSTWTM]

SET TXEN:
        RAMFILE_[PMA],                  ;Get adr of current header
         IF NOT CT [IZ] CALL [GET RX HDR]
        IF CT [UOVR] CALL [CLR XGO]     ;In case lost
RSTWTM: J[WRIO TMP]_RAMFILE ADR         ;Address tmp loc again
        RAMFILE_[W3],NRFRD/1,RETURN     ;Restore W3 and exit

.IF/FTTTYF
;Here because wrote XMT char word
WRIOT5: CALL [SET PCI TXEN]             ;Be sure xmtr going
        [W6]_Y,Y_RAMFILE ADR            ;Address status word
        RAMFILE_[W5],NRFRD/1,
         GOTO [CLR XDN]                 ;Clear transmit done int flag
.ENDIF/FTTTYF

;Here to write TTYRCV BITS
WRIOTM: [MEM-OP]_Y,ALU_RAMFILE,O_UOVR,
         RETURN
```

```
BSIO:   [AC-OP].OR.[MEM-OP]_B,          ;Add new bits we want
         GOTOP [WRIO.1]

;BSIOB:


BCIO:   [AC-OP]BAR&[MEM-OP]_B,          ;Clear bits we don't like
         GOTOP [WRIO.1]

;BCIOB:
```

```
.TOC    "Trap Handling"

;Here to store nothing
TO NOWHERE:
        SET LOCAL,
         IF [NOT TRAP] THENP [IFETCH]
        ;GOTOP [TRAP]

;Here at end of instruction to see if need to trap
TRAP:   J[0600]_[W1],CLEAR TRAP          ;Trap flags
        SWAP [W1]_[PMA],SET LOCAL,
         IF [NOT PAGED] THENP [IFETCH]
        [PC FLAGS]&[PMA]_B,              ;Check for need to do a trap
         IF CT [IZ] THENP [IFETCH]
        PUSH J[10.]_CTR
        ROL [PMA]_[PMA],RFCT
        J[420]_[W1]
        [W1]+[PMA]_B,
         CALL [U!E READ]
        GOTO [IFETCH 4]                  ;Do instruction

U!E READ:
        IF [EXEC] THEN [U!E R2]
        [UPT]+[PMA]_B,MEM START READ,
         GOTO [UNMAPPED READ]
U!E R2: [EPT]+[PMA]_B,MEM START READ,
         GOTO [UNMAPPED READ]

SET OV!T1:                               ;Here to set overflow and trap 1
        J[4002]_[W1],SET TRAP
        SWAP [W1]_[W1],
         GOTO [64*W1!PCF]

SET TRAP 1:                              ;Set trap 1
        J[200]_[W1],SET TRAP             ;Bit 28
        SWAP [W1]_[W1],                  ;Becomes a bit 10
         GOTO [W1!PCF]

SET TRAP 2:                              ;Set trap 2
        J[400]_[W1],SET TRAP             ;Bit 27
        SWAP [W1]_[W1],                  ;Becomes bit 9
         GOTO [W1!PCF]

64*W1!PCF:
        4*[W1]_B,J[1]_CTR                ;Shift bits left
SPCF 2: 4*[W1]_B,LOOP [SPCF 2]           ;Keep shifting bits
W1!PCF: [W1].OR.[PC FLAGS]_B,
         RETURN
```

```
.TOC    "INTERRUPT HANDLING"

;If this is a clock interrupt will return, but may clobber W2-W3
; If not clock will return to IFETCH
CHK INT:
        [K -1]+[TIME]_B,IX_MX,          ;Assume clock & tick it
         MEM HOLD,                      ; Don't mess up memory
         GRANT INTERRUPT,               ; Grant interrupt
         IF [NOT TIMER FLAG] THENP [SKIP CHAIN]

;Here for an interrupt on level 0 ... clock
        [PI REG]_Y,Y_2914 STATUS,
         MEM HOLD,
         IF NOT CT [MZ] RETURN

;Here once/millisecond
        J[CLK W2]_RAMFILE ADR
        [W2]_RAMFILE
        J[CLK W3]_RAMFILE ADR
        [W3]_RAMFILE
        J[27.]_[TIME]                   ;Restart tick counter
        J[TIME BASE+1]_RAMFILE ADR
        J[4000.]_[W2]                   ;Will add four milliseconds
        [W2]_Q RAMFILE_[W2]             ;Get timebase from ramfile
        Q+[W2]_F,F_Q_Y,ALU_RAMFILE,     ;Update time base in ramfile
         IF NOT CT [IOVR] THEN [LOI2]
        JS[0000]_[W2]
        Q+[W2]_Y,ALU_RAMFILE            ;With positive sign
        J[TIME BASE]_RAMFILE ADR        ;Address high order part of count
        RAMFILE_[W2],NRFRD/1
        [W2]+1_Y,ALU_RAMFILE
LOI2:   J[TIME INTERVAL]_RAMFILE ADR
        RAMFILE_[W2],NRFRD/1,IX_MX
        J[INTERVAL COUNTER]_RAMFILE ADR ;Address the interval counter
        [W2]_Q RAMFILE_[W2],NRFRD/1,    ;Get the interval counter
         IF CT [MZ] THEN [LOI5]         ; No interval = no counter
        [K -1]+[W2]_Y,ALU_RAMFILE,      ;Count this interval
         IF NOT CT [IZ] THEN [LOI5]
        Q_Y,ALU_RAMFILE                 ;Reinitialize interval counter
        J[APR FLAGS]_RAMFILE ADR        ;Need to set interval flag in APR
        J[40]_[W2]                      ;Bit for interval
        [W2]_Q RAMFILE_[W2]
        Q.OR.[W2]_[W2],ALU_RAMFILE      ;Save updated APR FLAGS
        SWAP [W2]_[W3]                  ;Get enables
        [W2]&[W3]_Y,IX_MX               ;Check for flag was enabled
        J[7]_[W3]                       ;Mask for PI level
        [W2]&[W3]_B,                    ;Check for Zero PI level
         IF CT [MZ IX_MX] THEN [LOI5]   ; If not enabled done
        J[BIT28]_[W2]                   ;Base to index into bit table
        [W2]+[W3]_Y,Y_RAMFILE ADR,      ;Address bit
         IF CT [MZ] THEN [LOI5]         ; If PI level=0 done
        RAMFILE_[W3],NRFRD/1            ;Get bit
        J[RF PROC REG]_RAMFILE ADR      ;Address last value of PROC REG
        RAMFILE_[W2],NRFRD/1            ;Get last value of processor reg
        [W2].OR.[W3]_Y,ALU_RAMFILE      ;Save new last value of PROC REG
        [W2].OR.[W3]_Y,ALU_PROC REG     ;Change software interrupts
LOI5:   J[CLK W2]_RAMFILE ADR           ;Where we saved W2
        RAMFILE_[W2],NRFRD/1
        J[CLK W3]_RAMFILE ADR           ;Where we saved W3
```

```
RAMFILE_[W3],NRFRD/1
[PMA]_[PMA],MEM START READ,      ;Restart memory read
 RETURN
```

```
;Here if interrupt is not clock
SKIP CHAIN:
        [TIME]+1_[TIME],                ;Correct time register
         IF [NOT RUN] THENP [SKPC2]
        [K -1]+[PC RH]_B,POP            ;Back up PC
SKPC2:  SET LOCAL,[PI REG]_[PXCT],POP   ;Save previous level
        2914 STATUS_Y,Y_[PI REG],POP    ;Get PI REG in case of int
        J[7]_[W6],POP                   ;In case this is PI lvl 1 int
.IF/DEBUG3
        [W6]&[PXCT]_B
FOOEY:  [W6].XOR.[PXCT]_Y,
         IF CT [IZ] THEN [FOOEY]
.ENDIF/DEBUG3
        [W6]&[PI REG]_B,POP             ;Clear extraneous bits
         IF [PXCT] CALL [SET CURRENT CTXT]
        [W6].XOR.[PI REG]_Y,            ;Check level
         IF NOT CT [IZ] THEN [SKPC8]    ; If not level 1 not PCI
.IF/DEBUG3
        J[POOP FLAG]_RAMFILE ADR
        RAMFILE_[W1],NRFRD/1
POOP:   [W1]_[W1],
         IF NOT CT [IZ] THEN [POOP]
        ONES_F,F_Y,ALU_RAMFILE
.ENDIF/DEBUG3
        LINE #_[W1],POP                 ;In case this is a line int
;Contents of W1 (line # reg) are
;       bits 33-35 = line number
;       bit 31-32 = code, 0=receive, 1=transmit, 2=dschange
;       bit 30 = 0
;       bit 29 = invalid
;       bits 0-28 = garbage
        J[100]_[W2]                     ;Mask for invalid flag
SKPC5:  [W1]&[W2]_Y,                    ;Check for invalid flag
         IF NOT CT [IZ] THEN [SKPC5]    ; If not valid chase skip chain
        [W1]&[W6]_B                     ;Leave only line number
        SWAP [W6]_[W6 LH]               ;Put line number in LH also
        2*([W6]+[W6 RH])_B              ;Makes 4*line# in W6 RH
        [W6]+[W6 RH]_F,                 ;Make 10*line#
         F_Q FJ[LNOSW]_[W6 RH]          ; adr of first line block
        Q+[W6]_[W6 RH],                 ;Leaves line#,,line block adr
         Y_RAMFILE ADR
        RAMFILE_[W5],NRFRD/1            ;Get status word for line
        J[10]_[W2]                      ;Mask for dispatch
        [W1]&[W2]_Y,                    ;Check for transmit
         IF NOT CT [IZ] THENP [INT XMT]
        J[20]_[W2]                      ;Another mask
        [W1]&[W2]_Y,                    ;Check for rcv or dsc
         IF CT [IZ] THENP [INT RCV]
        GOTOP [INT DSC]
```

```
;Here if interrupt is not for a PCI line
SKPC8:  J[BIT36]_[W1]                   ;Bit table
        [W1]-[PI REG]_Y,Y_RAMFILE ADR,  ;Address bit for level
         CALL [RAMFILE_W1]              ; Get bit for level
        J[PI IN PROG]_RAMFILE ADR
        [W1]_Q RAMFILE_[W1],NRFRD/1
        Q.OR.[W1]_Y,ALU_RAMFILE         ;Add this level also in progress
        J[60]_[E]                       ;New PC is 40+2n
        [E]-[PI REG]_Q,
         J[23B MASK]_RAMFILE ADR        ; virtual addresses are 23 bits
        Q-[PI REG]_F,RAMFILE_Y,NRFRD/1,
         F_Q Y_[W5],IF [TOPS20] THEN [SKPC82]
        COMPLEMENT [K -1.0]_F,F_[W5]    ;Smaller address for TOPS10
SKPC82: J[INT HALT]_[W1]                ;In case we get a page fail
        CALL [SET PF RCOVR]
        Q+[EPT]_[PMA],MEM START READ,   ;Pick up interrupt instruction
         CALL [UNMAPPED READ]
        [MB]_[E],IX_MX                  ;This is adr of XPCW block
        [W5]BAR&[MB]_Y,
         IF CT [MZ IX_MX] THEN [INT HALT] ; If location is 0 stop
        [PC FLAGS]_[MB]                 ;Save program flags
        ROR [BIT4]_[W1],                ;Clear user mode
         CALL [W1BAR&FLAGS]
        ROR [W1]_[W1],                  ;Make a bit 6 = prev ctx user
         CALL [W1BAR&FLAGS]
        IF [USER] CALL [W1!PC!]         ;Set prev ctxt user
        [K 7777.-1]&[PC]_B,SET EXEC,    ;References are now exec
         IF CT [MZ] THEN [SKPC9]        ; Check high order bits
.IF/FT1OPAG
        J[0264]_[W1]                    ;JSR instruction
        SWAP [W1]_[W1],CALL [W1 LLS9]   ;Make a JSR instruction
        [K -1.0]&[PC]_Y,IX_MX,          ;Check for illegal PC
         IF [TOPS20] THEN [INT HALT]    ; JSR is only for TOPS10
        [W1].XOR.[E]_F,F_B,             ;Check opcode
         IF NOT CT [MZ] THEN [INT HALT] ; Check illegal PC
        [K -1.0]&[E]_Y,                 ;Check if it was a JSR
         IF NOT CT [IZ] THEN [INT HALT]
        J[PUSHJ FLAGS]_RAMFILE ADR      ;Flags to clear on a JSR
        RAMFILE_[W1],NRFRD/1,           ;Get flags to clear
         CALL [W1BAR&FLAGS]
        [PC]_[MB RH],                   ;Add PC
         CALL [MEMORY WRITE]            ;Do JSR
        ZERO_[W1],
         CALL [SET PF RCOVR]
        [E]+1_[PC RH],SET LOCAL,        ;Set new PC
         GOTOP [IFETCH]
.ENDIF/FT1OPAG

INT HALT:
        J[HALT 111]_[W1]                ;Illegal interrupt instruction
        GOTOP [SET HALT CODE]

W1BAR&FLAGS:
        [W1]BAR&[PC FLAGS]_B,
         RETURN

;Here because interrupt instruction is a 23bit virtual address
SKPC9:  [W1]_[W5],                      ;Save bit6
```

```
         CALL [MEMORY WRITE]            ;Save flags in block
        [PC]_[MB],SET GLOBAL,           ;Save PC in block
         CALL [WRITE NEXT]
        [E]+1_Q_[E],                    ;Read new flags
         SPEC SEL/PAGE TABLE ENTRY,
         CALL [MEMORY READ]
        ROR [BIT4]_[W1],                ;Make a user mode bit
         CALL [W1BAR&MB_B]
        [W5]BAR&[MB]_B                  ;Clear prev ctxt user
        [W5]&[PC FLAGS]_B               ;Leave only previous context user
        [MB].OR.[PC FLAGS]_B,           ;Set new flags
         CALL [READ NEXT]               ;Read new PC
        ZERO_[W1],                      ;Done with special
         CALL [SET PF RCOVR]            ; Page fail recovery
        [MB]_[PC],SET LOCAL,            ;Set new PC
         GOTOP [IFETCH]

W1BAR&MB_B:
        [W1]BAR&[MB]_B,                 ;Force exec mode
         RETURN
```

```
.TOC    "STORE INSTRUCTION RESULTS"

;Store one word in memory, and two AC's (MULB, DIVB)
TO MEM D AC:
        [MEM-OP]_[MB],
         IF [NOT PXCT] THEN [TMDAC4]
        CALL [PXCT STORE 200]
        GOTO [D TO AC.0]
TMDAC4: [K 7777.-1]&[E]_F,F_Q_B,1_UC,    ;Address paging ram
         SPEC SEL/PAGE TABLE ENTRY,
         CALL [MEM WRITE 1]
        ;GOTO [D TO AC.0]

;Here to store two words in AC's SEL AC+[0] nc   iready selected
D TO AC.0:
        SEL AC+[1]                       ;Point ramfile at AC+1
        [MEM-OP+1]_RAMFILE,              ;Store AC+1
         GOTO [TO AC.0]

;Store a word in an AC
TO AC.0:
        SEL AC+[0]                       ;Point ramfile at AC

;Store a word in an AC, AC already selected
TO AC:  [MEM-OP]_RAMFILE,SET LOCAL,      ;Write AC
         IF [NOT TRAP] THENP [IFETCH]
        GOTOP [TRAP]

;Here to store two words in AC's SEL AC+[0] already selected
D TO AC:
        [MEM-OP]_RAMFILE,SEL AC+[1]      ;Save first half of result
        [MEM-OP+1]_RAMFILE,SET LOCAL,    ;Save second operand
         IF [NOT TRAP] THEN [IFETCH]
        GOTOP [TRAP]

;Store a word in an AC, AC already selected
AC TO AC:
        [AC-OP]_RAMFILE,                 ;Write AC
         SET LOCAL,
         IF [NOT TRAP] THENP [IFETCH]
        GOTOP [TRAP]

;Here to store to self after having read location
BACK TO SELF:
.IF/FAST
        [PMA]_[PMA],MEM START WRITE,
         IF [AC REF] THEN [TO SELF]
        [MEM-OP]_MEM,                    ;Write data in memory
         IF [AC.EQ.0] THEN [BACK TO MEM 2]
        [MEM-OP]_[MB],ALU_RAMFILE,       ;Write data in AC
         MEM HOLD,
         IF [NO BUS ERROR] THEN [TO NOWHERE]
        ;GOTO [TO SELF]                  ;Try again
.ENDIF/FAST
;Store to self, AC already selected
TO SELF:
        IF [AC.EQ.0] THEN [TO MEM]       ;IF AC is zero store mem, else both
.IF/FAST
        [MEM-OP]_RAMFILE,                ;Store AC
```

```
         GOTO [TO MEM]
.ENDIF/FAST

;Store to both
TO BOTH.O:
        SEL AC+[0]                      ;Point ramfile at AC
;Store to both, AC already selected
TO BOTH:
        [MEM-OP]_RAMFILE,               ;Store AC
         GOTO [TO MEM]

;Store memory data in AC and AC data in memory (EXCH instruction)
TO EACH:
        [MEM-OP]_RAMFILE                ;Write AC
        [AC-OP]_[MEM-OP]
        ;GOTO [BACK TO MEM]

;Here when fetched word from memory and want to write back updated number
BACK TO MEM:
.IF/FAST
        [PMA]_[PMA],MEM START WRITE,
         IF [NOT AC REF] THEN [BACK TO MEM 1]
        [PMA]_Y,SPEC SEL/VMA,           ;Address AC
         IF [PXCT] THEN [TO MEM]
        [MEM-OP]_RAMFILE,SET LOCAL,
         IF [NOT TRAP] THENP [IFETCH]
        GOTO [TRAP]
BACK TO MEM 1:
        [MEM-OP]_MEM                    ;Write data
BACK TO MEM 2:
        MEM HOLD,SET LOCAL,
         IF [NO BUS ERROR] THEN [TO NOWHERE]
        ;GOTO [TO MEM]                  ;Try again
.ENDIF/FAST
;Store AC to memory
TO MEM: [MEM-OP]_[MB],                  ;Put data in MB
         IF [PXCT] THEN [TO MEM PXCT]
        [K 7777.-1]&[E]_F,F_Q_B,1_UC,   ;Address paging ram
         SPEC SEL/PAGE TABLE ENTRY,
         CALL [MEM WRITE 1]
.IF/FAST
        SET LOCAL,
         IF [NOT TRAP] THEN [IFETCH]
        GOTO [TRAP]
.ENDIF/FAST
.IFNOT/FAST
        GOTO [TO NOWHERE]
.ENDIF/FAST
TO MEM PXCT:
        CALL [PXCT STORE 200]           ;Store data
        GOTOP [TO NOWHERE]

;Store 2 words to memory
D TO MEM:
        [MEM-OP]_[MB],                  ;Put date in right reg
         IF [PXCT] THEN [D TO MEM PXCT]
        [K 7777.-1]&[E]_F,F_Q_B,1_UC,   ;Address paging ram
         SPEC SEL/PAGE TABLE ENTRY,
         CALL [MEM WRITE 1]
        [MEM-OP+1]_[MB],                ;Copy rest of data to memory
```

```
         CALL [WRITE NEXT]
.IF/FAST
        SET LOCAL,
         IF [NOT TRAP] THEN [IFETCH]
        GOTO [TRAP]
.ENDIF/FAST
.IFNOT/FAST
        GOTO [TO NOWHERE]
.ENDIF/FAST
D TO MEM PXCT:
        J[0200]_[PXCT RH]
        CALL [SET PXCT CTXT]
        [K 7777.-1]&[E]_F,F_Q_B,1_UC,    ;Address paging ram
         SPEC SEL/PAGE TABLE ENTRY,
         CALL [MEM WRITE 1]
        [MEM-OP+1]_[MB],                 ;Copy rest of data to memory
         CALL [WRITE NEXT]
        CALL [SET CURRENT CTXT]
        GOTOP [TO NOWHERE]

;Here to store 4 words in ACs
Q TO AC.0:
        SEL AC+[0]
        [MEM-OP]_RAMFILE,
         SEL AC+[1]
        [MEM-OP+1]_RAMFILE,
         SEL AC+[2]
        [AC-OP]_RAMFILE,
         SEL AC+[3]
        [AC-OP+1]_RAMFILE,
         SET LOCAL,
         IF [NOT TRAP] THENP [IFETCH]
        GOTOP [TRAP]
```

```
;Here to clear page table
PAGE TABLE CLEAR:
        J[511.]_CTR,                    ;Want to clear 512 entries
         J[PAGE TABLE-1]_[W1]
CLR RAM:
        [W1]+1_[W1],Y_RAMFILE ADR       ;Address next page table entry
        ZERO_Y,ALU_RAMFILE,
         LOOP [CLR RAM]
        RETURN
```

```
.TOC    "TOPS20 Page Refill"
;TOPS20 page refill logic
; Call: x&[K 7777.-1]_[E],      ;Virtual adr to do page refill for
;        SPEC SEL/PAGE TABLE ENTRY      ;Set section flags
;       1_UC,CALL [PAGE W REFIL]        ;To do refill for a write reference
;       0_UC,CALL [PAGE R REFIL]        ;To do refill for a read reference
;
; flag usage:
;       uc = T
;       uz = section pointer
;
; reg usage:
;       W1 = Virtual section number
;       W3 = AND of all section and page pointers (for W)

PAGE W REFIL:
        J[MB REFIL]_RAMFILE ADR,1_UC    ;Address location to save MB
        [MB]_RAMFILE,                   ;Save MB
         GOTO [PR 1]

PAGE R REFIL:
        ; 0_UC
PR 1:
.IF/FT2OPAG
.IF/FT1OPAG
        IF [TOPS1O] THEN [T1OPR]
.ENDIF/FT1OPAG
        SWAP [E]_[PMA],                 ;Put section number in RH
         IF [ILLEGAL SECTION] THENP [PF X27]
        [K 77]&[PMA]_B,                 ;Strip LH
         IF [EXEC] THEN [PR 2]
        [UPT]+[PMA]_B,GOTO [PR 3]       ;UPT + section # to PMA
PR 2:   [EPT]+[PMA]_B                   ;EPT + section # to PMA
PR 3:   J[ESECT]_[W1]                   ;Get offset for section pointers
        [BIT4]_[W3],0_UZ                ;W bit(Mask for AND of pointers)
        [W1]+[PMA]_F,F_Q_B,             ;EPT(UPT)+sect+ESECT
         MEM START READ,
         CALL [PTR EVAL]                ; Get page adr from section table
        [MB]_[W2],                      ;Save page number
         CALL [CST UPDATE]              ;Update CST for page table
        2*[E]_[W1],PUSH J[3]_CTR        ;Copy virtual address to
        4*[W1]_B,RFCT                   ;Convert to page number
        SWAP [W1]_[PMA]                 ;Right justifty page number
        [K 777]&[PMA]_B                 ;Make in section page number
        [W4]+[PMA]_B,MEM START READ,    ;Address page table entry
         CALL [PTR EVAL]                ; and fetch pointer
        [MB]_[W2],1_UZ,                 ;Update M
         CALL [CST UPDATE]              ; Update CST for page
        J[1]_[W1]                       ;Mask for M bit
        [MB]&[W1]_Y,IX_MX               ;Latch M as 0_MZ
        2*[W3]_Q,                       ;Make a bit 4 if writable
         IF CT [MZ] THEN [PR 6]
        Q.OR.[W3]_[W3]                  ;Add M bit if W
PR 6:   J[BIT2]_RAMFILE ADR,            ;Bit for valid mapping
         [W3].OR.[W4]_Q                 ;Add W bit
        RAMFILE_[W1],NRFRD/1            ;Get valid bit from ramfile
        Q.OR.[W1]_[MB],                 ;Add valid bit to paging ram word
         IF [USER] CALL [BO!MB]
        [E]_Y,                          ;Address paging ram location
```

```
         SPEC SEL/PAGE TABLE ENTRY
.IF/DEBUGPF
        JS[1776]_[W2]
        ROR [W2]_[W2]
        [MB]&[W2]_Y,
         IF NOT CT [IZ] THEN [HALT PG]
.ENDIF/DEBUGPF
        [MB]_[W1],ALU_RAMFILE,          ;Fill paging ram
         IF NOT CT [UC] RETURN          ; If read refill we are done
        J[MB REFIL]_RAMFILE ADR
        RAMFILE_[MB],NRFRD/1,
         RETURN
```

```
;Here to evaluate a section pointer or map pointer
; call: x_Q_[PMA],       ;Physical address of 1st pointer
;        MEM START READ,
;        CALL [PTR EVAL]
; Returns
;       accumulates W in W3
;       page number in MB (maybe also other junk)
;       address in W4
PTR EVAL:
        MEM HOLD,                          ;Get pointer from memory
         CALL [UNMAPPED READ]
        [MB]&[W3]_B                        ;Accumulate W
        2*[MB]_[W4],IX_MX                  ;Latch bit0 as MN
        2*[W4]_[W4],                       ;Copy pointer to W4 and shift it
         IF CT [MN IX_MX] THEN [PF X0]     ; for decode on bits 0,1,2
        2*[W4]_[W4],
         IF NOT CT [MN IX_MX] THEN [PTE 7]      ;Branch if bits 00x

;Here if bits 0-2 of section pointer are 01x
        [K -1.0]BAR&[MB]_Q,                ;Get SPT index
         IF NOT CT [MN] THEN [PTE 1]       ;branch for  010

;Here if bits 0-2 of section pointer are 011 = indirect pointer
        SWAP [MB]_[W1]                     ;Put offset in RH
        [K 777]&[W1]_B,                    ;Mask off extra bits
         CALL [PTE 1]                      ;Get table address from SPT
        [W4]_[PMA]                         ;Address of indirect table
        [W1]+[PMA]_B,MEM START READ,
         GOTO [PTR EVAL]

;Here if bits 0-2 of section pointer are 010 = shared pointer
PTE 1:  Q+[SPT]_F,F_Q_[PMA],               ;Get SPT entry for page map
         MEM START READ,
         CALL [UNMAPPED READ]
        ;This word isn't included in the W accumulation
        2*[MB]_[W4],1_MX,                  ;Copy page address to W4
         CALL [4*W4_B]
```

```
;Here if bits 0-2 of pointer are 00x
PTE 7:  4*[W4]_B,
         IF NOT CT [MN] THEN [PF XO]

;Here if bits 0-2 of pointer are 001 = immediate pointer
PTE 8:  SWAP [K 77]_[W1]                ;Mask for storage medium
        [MB]&[W1]_Y,IX_MX               ;See if in core or else where
        [E]_[W4 LH],                    ;Add virtual section
         IF NOT CT [MZ] THEN [PF XO]
        4*[W4]_B                        ;Shift page number to make adr
4*W4_B: 4*[W4]_B,
         RETURN
```

```
;Here to update a CST entry
; Call: x_[MB]          ;Physical page number for update
;       Call with UC if write reference
;       Call with UZ if want M updated
CST UPDATE:
        J[CST]_RAMFILE ADR
        J[7777]_[PMA]
        [K -1]&[PMA]_F,LLS F_B,         ;Leave 17777 in PMA
         1_S100 1_Q100
        [MB]&[PMA]_F,                   ;Leave only page number in Q
         F_Q RAMFILE_[PMA]              ; Get CST adr in PMA
        Q+[PMA]_[PMA],MEM START READ,   ;Get CST entry for page map
         CALL [UNMAPPED READ]
        J[CSTMSK]_RAMFILE ADR
        [K 7777.-1]BAR&[MB]_Y,IX_MX     ;Check age field
        [MB]_Q RAMFILE_[MB],            ;Get CSTmask register
         IF CT [MZ] THEN [AGE PF]       ; If age trap page fail
        Q&[MB]_Q,                       ;Mask off bits
         J[CSTDATA]_RAMFILE ADR
        RAMFILE_[MB],NRFRD/1
        Q.OR.[MB]_F,F_Q_[MB],           ;Add new bits
         IF NOT CT [UC] THEN [CSTUD8]   ; Branch if read refill
        [W3]_Y,                         ;Test W bit
         IF NOT CT [UZ IX_MX] THEN [CSTUD8]
        J[1]_[MB]                       ;Get M bit for CST entry
        Q.OR.[MB]_[MB],                 ;Set M bit
         IF CT [MZ] THEN [PF XO]        ; If not W, page fail
CSTUD8: [PMA]_[PMA],MEM START WRITE,    ;Write updated CST entry
         GOTO [UNMAPPED WRITE]
.ENDIF/FT2OPAG
```

```
.TOC    "TOPS10 Page Refill"

.IF/FT1OPAG
T1OPR:  [K -1.0]&[E]_Y,                 ;Check for legal section
         IF NOT CT [IZ] THENP [PF X27]
        [K 777]BAR&[E]_F,F_Q            ;Strip onpage portion
        Q_[W1],PUSH J[3]_CTR            ;Shift E left 8 places
        4*[W1]_B,RFCT
        SWAP [W1]_[PMA],                ;Sign bit = E bit 26
         IF [USER] THEN [T1PR14]        ; Page #/2 in bits 28-35
        B SEL/E,
         IF [B BIT 18] THEN [T1PR13]
        ROL [PMA]_[W2]                  ;Page # goes to bits 27-35
        J[340]_[W1]
        [W2]-[W1]_Y,                    ;Check for up to 340000
         IF CT [IN] THEN [T1PR12]
        J[220]_[W1]                     ;Offset for 340000 in UPT
        [W1]+[PMA]_B,
         GOTO [T1PR14]
T1PR12: J[600]_[W1]                     ;Offset for 000000 in exec
        [W1]+[PMA]_B
T1PR13: [EPT]+[PMA]_B,MEM START READ,   ;Offset for 400000 in EPT
         CALL [UNMAPPED READ]           ;Get word from EPT
        GOTO [T1PR15]
T1PR14: [UPT]+[PMA]_B,MEM START READ,   ;Get word from UPT
         CALL [UNMAPPED READ]
T1PR15: [PMA]_Y,IX_MX                   ;Check for odd or even
        IF NOT CT [MN] CALL [SWAP MB_MB]
        J[7777]_[W2]                    ;Want mask for physical page
        [W2]_F,LLS F_B,B SEL/W2,        ;Put 17777 in W2
         1_SIOO 1_QIOO
        ZERO_[W1],                      ;Build paging ram word here
         IF [EXEC] THEN [T1PR24]
        J[0002]_[W1]                    ;Bit for user
        SWAP [W1]_[W1]
T1PR24: [MB]_[W1 RH]                    ;Add APWS
        2*[W1]_[W1]                     ;Put A in bit17
        2*[W1]_[W1 RH]                  ;Put W in bit18(discard P)
        4*[W1]_B                        ;Put S in bit17
        [MB]_[W1 RH]                    ;Add page #
        [W2]&[W1 RH]_B,PUSH J[1]_CTR    ;Clear extra bits
        [W1]+[W1 LH]_F,LLS F_B,RFCT     ;Put S in bit 13
        [E]_Y,SPEC SEL/PAGE TABLE ENTRY ;Address paging ram
        2*[W1]_[W1],PUSH J[3]_CTR       ;Want to shift left 9
        4*[W1]_B,IX_MX,RFCT
        [W1]_Y,ALU_RAMFILE,             ;Write new paging ram word
         IF NOT CT [UC] RETURN
        J[MB REFIL]_RAMFILE ADR         ;Address where we left MB
        RAMFILE_[MB],NRFRD/1,RETURN     ;Restore MB
.ENDIF/FT1OPAG
```

```
.TOC    "UNMAPPED MEMORY READ"

; CALL: x_[PMA],MEM START READ
;       returns with data in MB
UNMAPPED READ 0:
        [PMA]_[PMA].MEM START READ
UNMAPPED READ:
        MEM HOLD,
         CHECK INTERRUPTS
        MEM_[MB]                        ;Get contents of memory
        MEM HOLD,
         IF [NO BUS ERROR] RETURN
        ;GOTO [MEM ERR]
```

```
;Here on a mem fault
; If soft (ECC correctable error) will save status (even if already
;       latched) set APR flag, SPEC SEL/IW, and return.
;       Will clobber W1&W2
; If RAMFILE [ME RCOVR] is nonzero will go there with status in W1
; Else will build a page fail code and go to PF XX
MEM ERR:
        J[ME W1]_RAMFILE ADR            ;Have to save W1
        [W1]_RAMFILE
        J[ME W2]_RAMFILE ADR            ;Have to save W2
        [W2]_RAMFILE
        [PMA]_[W2],MEM HOLD,FREE/3      ;Copy offending address
        4*[W2]_B,MEM HOLD               ;Put board number in lh
        SWAP [W2]_[W2],MEM HOLD         ;Put board number in rh
        [K 77]&[W2]_Q,MEM HOLD,
         J[BIT20]_RAMFILE ADR
        RAMFILE_[W2],NRFRD/1,MEM HOLD   ;Get 100000
        Q+[W2]_F,F_Q_Y,START IO READ    ;Address memory status reg
        IO TRANSFER,XFER MEM,           ;Get status from memory
         Y_[W1],ALU_Y/NO,
         M STATUS ENAB/NO               ;Don't change error flags
        [BIT17]_[W1 LH],                ;Check for NXM
         IF [NOT MEM EXISTS] THEN [MEM NXM]
        [BIT17]+[W1]_B,                 ;1st check Bus fault line
         IF [MEM FAULT] THEN [MEM FAULT]

;Here if we detected parity from bus
 IF/FTCKBP
        [BIT17]+[W1]_B,                 ;Must be a bus parity error
         GOTO [MERO5]
.ENDIF/FTCKBP
.IFNOT/FTCKBP
        J[ME W1]_RAMFILE ADR            ;Where we saved W1
        RAMFILE_[W1]
        J[ME W2]_RAMFILE ADR            ;Where we saved W2
        RAMFILE_[W2]
        [MB]_[MB],SPEC SEL/IW,          ;In case part of efa calc
         RETURN
.ENDIF/FTCKBP

;Here because we won a NXM
MEM NXM:
        J[NXM STS]_[W2],Y_RAMFILE ADR   ;Save status as "nxm"
        GOTO [MER10]

;Here when memory complains
MEM FAULT:
.IF/FTSMER
        J[6550]_[W2]
        4*[W2]_B,CALL [4*W2_B]
        [W1]&[W2]_Q                     ;Leave only bits we care about
        J[4010]_[W2]                    ;Bits we expect
        4*[W2]_B,CALL [4*W2_B]
        Q.XOR.[W2]_F,F_Y,
         IF NOT CT [IZ] THEN [MERO5]
        J[APR FLAGS]_RAMFILE ADR
        [K 77]+1_F,F_Q Y_[W2],          ;Make 100 = SME flag
         RAMFILE_Y,NRFRD/1
```

```
        Q.OR.[W2]_F,F_Q_[W2],
         ALU_RAMFILE
        SWAP [W2]_[W2]
        Q&[W2]_Y,
         IF CT [IZ] THEN [MERO4]
        **** add code to make interrupt ****
MERO4:  J[SME STS]_[W2],Y_RAMFILE ADR   ;Save status as "soft"
         CALL [MESAV2]                  ;Save status and exit
        J[ME W1]_RAMFILE ADR
        RAMFILE_[W1]
        J[ME W2]_RAMFILE ADR
        RAMFILE_[W2]
        [MB]_Y,SPEC SEL/IW,
         RETURN
.ENDIF/FTSMER
MERO5:  J[HME STS]_[W2],Y_RAMFILE ADR,  ;Save status as "hard"
         POP
MER10:  J[ME RCOVR]_RAMFILE ADR,        ;Check for recovery routine
         [W2]_Q,POP
        RAMFILE_[W2],NRFRD/1,POP        ;Get recovery routine
        [W2]_Y,IF NOT CT [IZ] THEN Y    ;If there is one do it
.IFNOT/FTPFRMER
        J[PF RCOVR]_RAMFILE ADR ;Page fail recovery
        ZERO_Y,ALU_RAMFILE              ;  doesn't help here
.ENDIF/FTPFRMER
        Q_[W2],Y_RAMFILE ADR,
         CALL [MERSAV]                  ;Save error status
        [BIT17].XOR.[W1]_F,POP,
         FJ[3700]_[W1]                  ;Code for NXM
        [K -1.0]&[W1]_Y,
         IF CT [IZ] THENP [PF XX]
        J[3600]_[W1]                    ;Code for mem err
        GOTOP [PF XX]                   ;Perform a page fail

MERSAV: RAMFILE_Y,NRFRD/1,IX_MX         ;See if already latched error
        IF NOT CT [MZ] RETURN           ;Don't overwrite
MESAV2: [W1]_RAMFILE,                   ;Save status
         CALL [W2+1_RFA]
        [PMA]_RAMFILE,                  ;Save address of losing location
         CALL [W2+1_RFA]
        [MB]_RAMFILE,SPEC SEL/IW,
         RETURN

4*W2_B: 4*[W2]_B,RETURN

W2+1_RFA:
        [W2]+1_[W2],Y_RAMFILE ADR,      ;Address next ramfile location
         RETURN
```

```
.TOC    "MEMORY READ ROUTINE"

; CALL: x_Q_[E],                          ;Put address in E & Q
;        CLEAR LOCAL,SET LOCAL,           ; Only one please
;        SPEC SEL/PAGE TABLE ENTRY,       ; Address paging entry
;        CALL [MEMORY READ]
; RETURNS       with data in MB and with SPEC SEL/IW set
;               paging RAM word is in W1
;               clobbers W2-W4 (if page refill)

;Here to fetch next virtual address
READ NEXT:
        [E]+1_Q_[E RH],
         IF [LOCAL] THEN [MEM READ 0]
        Q_[E],                           ;Address to read
         SPEC SEL/PAGE TABLE ENTRY,
         GOTO [MEMORY READ]

MEM READ 0:
        [K 7777.-1]&[E]_F,F_Q_B,         ;Address to read
         SPEC SEL/PAGE TABLE ENTRY,
         GOTO [MEMORY READ]

MEMORY READ:
        Q&[K 777]_F,RAMFILE_Y,NRFRD/1,   ;Put page entry in W1
         0_UC IX_MX,
         Y_[W1] F_Q,LCTXTM/1,            ; Put on page adr in Q
         IF [UNPAGED OR AC] THEN [M R 8]
 IF/DEBUGPF
        JS[1776]_[W2]
        ROR [W2]_[W2]
        [W1]&[W2]_Y,
         IF NOT CT [IZ] THEN [HALT PG]
.ENDIF/DEBUGPF
        Q.OR.[W1]_[PMA],MEM START READ,
         IF [CONTEXT MATCH] THEN [MEM READ 4]
        [K 7777.-1]&[E]_B,0_UC,          ;Mask address to 30 bits
         CALL [PAGE R REFIL]
        GOTO [MEM READ 0]

MEM READ 3:
        [E]_[PMA],MEM START READ         ;Address location
MEM READ 4:
        CHECK INTERRUPTS,
         IF [NO] HOLD
        MEM [MB],SPEC SEL/IW             ;Read memory into MB & select IW
        [MB]_[MB],MEM HOLD,
         IF [NO BUS ERROR] RETURN
        MEM HOLD,GOTO [MEM ERR]

;Here if reference is unpaged or ac
M R 8:  [E]_[PMA],SPEC SEL/VMA,          ;Address register
         IF [NOT AC REF] THEN [MEM READ 3]
        RAMFILE_[MB],NRFRD/1,            ;Read AC into MB & select IW
         IF [ILLEGAL SECTION] THENP [PF X27]
        CHECK INTERRUPTS
        [MB]_Y,SPEC SEL/IW,
         RETURN
```

```
.TOC    "UNMAPPED MEMORY WRITE ROUTINE"

; CALL: x_[MB]                          ;Put data in MB
;       x_[PMA]                         ;Physical location to write into
;       CALL [UNMAPPED WRITE]

;Here to write memory with contents of MB and then increment PMA
UNMAPPED WRITE NEXT:
        [PMA]+1_[PMA],MEM START WRITE,  ;Give memory address
         GOTO [UNMAPPED WRITE 2]

UNMAPPED WRITE:
        [PMA]_[PMA],                    ;Give memory address to write
         MEM START WRITE
UNMAPPED WRITE 2:
        [MB]_MEM                        ;Write data in memory
        MEM HOLD,
         IF [NO BUS ERROR] RETURN
        MEM HOLD,GOTO [MEM ERR]
```

```
.TOC    "MEMORY WRITE ROUTINE"

; CALL: x_[E]                           ;Virtual adr
        x_[MB],                         ;Put data in MB
;        CALL [MEMORY WRITE]
;        with GLOBAL/LOCAL flag set,
;       may clobber W1-W5

WRITE NEXT:
        [E]+1_F,F_Q_[E],
         SPEC SEL/PAGE TABLE ENTRY,
         GOTO [MEM WRITE 1]
MEMORY WRITE:
        [K 7777.-1]&[E]_F,F_Q_B,1_UC,   ;Address paging ram
         SPEC SEL/PAGE TABLE ENTRY,
         IF [NOT PAGED] THEN [M W 8]
MEM WRITE 1:
        Q&[K 777]_Q RAMFILE_[W1],NRFRD/1,       ;Get paging ram entry
         1_UC IX_MX,LCTXTM/1,
         IF [UNPAGED OR AC] THEN [M W 8]
.IF/DEBUGPF
        JS[1776]_[W2]
        ROR [W2]_[W2]
        [W1]&[W2]_Y,
         IF NOT CT [IZ] THEN [HALT PG]
.ENDIF/DEBUGPF
.IF/FTADRB
        J[AB WR]_RAMFILE ADR,           ;Get address to stop on
         Q&[K 7777.-1]_F,F_Q
        RAMFILE_[W2],NRFRD/1
        Q.XOR.[W2]_F,F_Y,
         IF CT [IZ] THEN [M W 2]
        J[HALT CODE]_RAMFILE ADR
        J[HALT ADRBRK]_[W2]
        [W2]_RAMFILE,CLEAR RUN
.ENDIF/FTADRB
M W 2:  Q.OR.[W1]_[PMA],
         MEM START WRITE,
         IF [CONTEXT MATCH] THEN [M W 3]
M W 25: [K 7777.-1]&[E]_B,              ;Mask address to 30 bits
         CALL [PAGE W REFIL]
        GOTO [MEMORY WRITE]

M W 3:  B SEL/W1,MEM HOLD,              ;Check M bit is set
         IF [B BIT 3] THEN [M W 5]
M W XX:
.IF/FT2OPAG
        [BIT4]&[W1]_Y,IX_MX,            ;Check W bit
         IF [TOPS10] THEN [M W PF]
        IF NOT CT [MZ] THEN [M W 25]
;***** clever code here could just call CST UPDATE
.ENDIF/FT2OPAG
M W PF:  GOTO [PAGE FAIL]

M W 4:  [E]_[PMA],MEM START WRITE
M W 5:  [MB]_MEM                        ;Write memory into MB
        MEM HOLD,
         IF [NO BUS ERROR] RETURN
        MEM HOLD,GOTO [MEM ERR]
```

```
;Here if writing AC or unpaged
M W 8:  [E]_[PMA],SPEC SEL/VMA,          ;Address register
         IF [NOT AC REF] THEN [M W 4]    ;Check for unpaged
        IF [ILLEGAL SECTION] THENP [PF X27]
        [MB]_RAMFILE,RETURN              ;Write AC in RAMFILE
```

```
.TOC    "PAGE FAIL"

.NOBIN
                        TOPS20                                  TOPS10
;           +-----------------------------------+   +-----------------------------------+
;       500 ! Page fail word                    !   ! Page fail word                    !
;           +-----------------------------------+   +-----------------------------------+
;       501 ! Page fail flags                   !   ! Page fail old flags,,PC           !
;           +-----------------------------------+   +-----------------------------------+
;       502 ! Page fail old PC                  !   ! Page fail new PC                  !
;           +-----------------------------------+   +-----------------------------------+
;       503 ! Page fail new PC                  !   ! Reserved                          !
;           +-----------------------------------+   +-----------------------------------+
.BIN

;Here for an ager page fail
AGE PF:
.IF/FTAGEPF
        J[2100]_[W1]
        GOTOP [PF XX]
.ENDIF/FTAGEPF

;Here for uncoded page fail
PF X0:  ZERO_[W1],
         GOTOP [PAGE FAIL]

;Here for an IO page fail
IO PF:
PF X20: J[2000]_[W1]
        GOTOP [PF XX]

;Here for a Illegal Indirect page fail
PF X24: J[2400]_[W1]
        GOTOP [PF XX]

;Here for an Illegal Section page fail
PF X27: J[2700]_[W1]
        ;GOTOP [PF XX]

PF XX:  SWAP [W1]_[W1],PUSH J[2]_CTR    ;Move code to lh
        4*[W1]_B,RFCT                   ;Shift 6 places left
        GOTO [PF XX1]

.IF/DEBUGPF
HALT PG:
        ONES_[W1],
         GOTO [SET HALT CODE]
.ENDIF/DEBUGPF

;Here to do a Page fail, with page fail word in W1
PAGE FAIL:
.IF/DEBUGPF
        js[1]_[w2]                      ;Mask for illegal bits
        ror [w2]_[w2]
        [w1]&[w2]_Y,
         if not ct [iz] then [halt pg]
.ENDIF/DEBUGPF
        ROR [BIT4]_[W2],                ;Make a type=1 bit
         IF NOT CT [UC] THEN [PF XX1]
```

```
        [W2].OR.[W1]_B                  ;Add T bit
PF XX1: J[37]_[W2]                      ;Build mask
        SWAP [W2]_[W2]
        [W2]&[E LH]_B,
         IF [PXCT] CALL [SET CURRENT CTXT]
        J[PF RCOVR]_RAMFILE ADR,
         [K 7777.-1]BAR&[W1]_Q          ;Strip mapping info
        Q.OR.[E]_[W1],                  ;Put virtual address in PF word
         IF [NOT PAGED] THENP [PF XX2]
        J[1000]_[W2]                    ;Will become V bit
        SWAP [W2]_[W2],
         CALL [W2.OR.W1]
PF XX2: [W1]_[MB],                      ;Copy page fail word
         IF [USER] CALL [BO!MB]         ; Add user mode if needed
        RAMFILE_[W2],!X_MX              ;Get recovery routine adr
        [W2]_Y,
         IF NOT CT [MZ] THEN Y          ;If there is one do it
        J[PF MER]_[W1]                  ;In case error fetching from UPT
        [K -1]+[PC RH]_B,               ;Backup the PC
         CALL [SET MER]                 ; Set memory err recovery
        J[PFW]_[PMA]                    ;Get offset into UPT for page fail word
        [UPT]+[PMA]_B,MEM START WRITE,  ;Physical location of word
         CALL [UNMAPPED WRITE]
        [PC FLAGS]_[MB],                ;Next word is flags
         IF [TOPS20] THEN [PF 34]
        [PC]_[MB RH],GOTO [PF 35]       ;Put PC in rh
PF 34:  [PMA]+1_[PMA],MEM START WRITE,  ;Store flags
         CALL [UNMAPPED WRITE]
        [PC]_[MB]                       ;Next word is PC
PF 35:  [PMA]+1_[PMA],MEM START WRITE,  ;Store PC
         CALL [UNMAPPED WRITE]
PF 40:  [PMA]+1_[PMA],MEM START READ,
         CALL [UNMAPPED READ]           ;Get new PC
        ZERO_[W1],CALL [SET MER]        ;Have now passed any errors
        [MB]_[PC],SET EXEC,
         IF [TOPS10] THEN [MB_PC FLAGS]
        ZERO_[PC FLAGS],SET LOCAL,
         GOTO [IFETCH]

BO!MB:  JS[0]_[W1]                      ;Put 400000,,0 in W1
W1!MB:  [W1].OR.[MB]_B,                 ;Set bit
         RETURN

;Here if encounter NXM, bus error, or ECC error
PF MER: J[HALT PFMER]_[W1]
        GOTOP [SET HALT CODE]           ;Set halt code and quit

SET PF ME RCOVR:
        CALL [SET MER]
        ;GOTO [SET PF RCOVR]
SET PF RCOVR:
        J[PF RCOVR]_RAMFILE ADR         ;In case errors handling int
W1_RAMFILE:
        [W1]_RAMFILE,
         RETURN

;Here to set memory error recovery location
SET MER:
        J[ME RCOVR]_RAMFILE ADR
        [W1]_Y,ALU_RAMFILE,
```

```
RETURN
```

```
.TOC    "DISPATCH TABLE"

        .DCODE

        [MUUO]#[APRID]          ;000 Illegal 70000
        [CHK PC SECT] [MCE] [LUUO]              ;001 LUUO 70004
        [CHK PC SECT] [MCE] [LUUO]              ;002 LUUO 70010
        [CHK PC SECT] [MCE] [LUUO]              ;003 LUUO 70014
        [CHK PC SECT] [MCE] [LUUO]#[CONO APR]   ;004 LUUO 70020
        [CHK PC SECT] [MCE] [LUUO]#[CONI APR]   ;005 LUUO 70024
        [CHK PC SECT] [MCE] [LUUO]#[CONSZ APR]  ;006 LUUO 70030
        [CHK PC SECT] [MCE] [LUUO]#[CONSO APR]  ;007 LUUO 70034
        [CHK PC SECT] [MCE] [LUUO]              ;010 LUUO 70040
        [CHK PC SECT] [MCE] [LUUO]              ;011 LUUO 70044
        [CHK PC SECT] [MCE] [LUUO]              ;012 LUUO 70050
        [CHK PC SECT] [MCE] [LUUO]              ;013 LUUO 70054
        [CHK PC SECT] [MCE] [LUUO]#[CONO PI]    ;014 LUUO 70060
        [CHK PC SECT] [MCE] [LUUO]#[CONI PI]    ;015 LUUO 70064
        [CHK PC SECT] [MCE] [LUUO]#[CONSZ PI]   ;016 LUUO 70070
        [CHK PC SECT] [MCE] [LUUO]#[CONSO PI]   ;017 LUUO 70074
        [CHK PC SECT] [MCE] [LUUO]              ;020 LUUO 70100
        [CHK PC SECT] [MCE] [LUUO]#[DATAI PAG]  ;021 LUUO 70104
        [CHK PC SECT] [MCE] [LUUO]#[CLRPT]      ;022 LUUO 70110
        [CHK PC SECT] [MCE] [LUUO]#[DATAO PAG]  ;023 LUUO 70114
        [CHK PC SECT] [MCE] [LUUO]#[CONO PAG]   ;024 LUUO 70120
        [CHK PC SECT] [MCE] [LUUO]#[CONI PAG]   ;025 LUUO 70124
        [CHK PC SECT] [MCE] [LUUO]              ;026 LUUO 70130
        [CHK PC SECT] [MCE] [LUUO]              ;027 LUUO 70134
        [CHK PC SECT] [MCE] [LUUO]              ;030 LUUO 70140
        [CHK PC SECT] [MCE] [LUUO]              ;031 LUUO 70144
        [CHK PC SECT] [MCE] [LUUO]              ;032 LUUO 70150
        [CHK PC SECT] [MCE] [LUUO]              ;033 LUUO 70154
        [CHK PC SECT] [MCE] [LUUO]              ;034 LUUO 70160
        [CHK PC SECT] [MCE] [LUUO]              ;035 LUUO 70164
        [CHK PC SECT] [MCE] [LUUO]              ;036 LUUO 7017C
        [CHK PC SECT] [MCE] [LUUO]              ;037 LUUO 70174
.IF/FT2OPAG
        [MUUO]#[RDSPB]          ;040 MUUO 70200
        [MUUO]#[RDCSB]          ;041 MUUO 70204
        [MUUO]#[RDPUR]          ;042 MUUO 70210
        [MUUO]#[RDCSTM]         ;043 MUUO 70214
.ENDIF/FT2OPAG
.IFNOT/FT2OPAG
        [MUUO]                  ;040 MUUO 70200
        [MUUO]                  ;041 MUUO 70204
        [MUUO]                  ;042 MUUO 70210
        [MUUO]                  ;043 MUUO 70214
.ENDIF/FT2OPAG
        [MUUO]#[RDTIM]          ;044 MUUO 70220
        [MUUO]#[RDINT]          ;045 MUUO 70224
        [MUUO]#[RDHSB]          ;046 MUUO 70230
        [MUUO]                  ;047 MUUO 70234
.IF/FT2OPAG
        [MUUO]#[WRSPB]          ;050 MUUO 70240
        [MUUO]#[WRCSB]          ;051 MUUO 70244
        [MUUO]#[WRPUR]          ;052 MUUO 70250
        [MUUO]#[WRCSTM]         ;053 MUUO 70254
.ENDIF/FT2OPAG
.IFNOT/FT2OPAG
```

```
        [MUUO]                          ;050 MUUO 70240
        [MUUO]                          ;051 MUUO 70244
        [MUUO]                          ;052 MUUO 70250
        [MUUO]                          ;053 MUUO 70254
ENDIF/FT2OPAG
        [MUUO]#[WRTIM]                  ;054 MUUO 70260
        [MUUO]#[WRINT]                  ;055 MUUO 70264
        [MUUO]#[WRHSB]                  ;056 MUUO 70270
        [MUUO]                          ;057 MUUO 70274
        [MUUO]                          ;060 MUUO 70300
        [MUUO]                          ;061 MUUO 70304
        [MUUO]                          ;062 MUUO 70310
        [MUUO]                          ;063 MUUO 70314
        [MUUO]                          ;064 MUUO 70320
        [MUUO]                          ;065 MUUO 70324
        [MUUO]                          ;066 MUUO 70330
        [MUUO]                          ;067 MUUO 70334
        [MUUO]                          ;070 MUUO 70340
        [MUUO]                          ;071 MUUO 70344
        [MUUO]                          ;072 MUUO 70350
        [MUUO]                          ;073 MUUO 70354
        [MUUO]                          ;074 MUUO 70360
        [MUUO]                          ;075 MUUO 70364
        [MUUO]                          ;076 MUUO 70370
        [MUUO]                          ;077 MUUO 70374
        [MUUO]                          ;100 70400
        [MUUO]                          ;101 70404
        [MUUO]                          ;102 70410
        [MUUO]                          ;103 70414
        [MUUO]                          ;104 70420
        [FETCH AC&I] [CHK PC SECT] [ADJSP] [ADJSP V]    ;105 ADJSP 70424
        [MUUO]                          ;106 70430
        [MUUO]                          ;107 70434
        [MUUO]                          ;110 DFAD 70440
        [MUUO]                          ;111 DFSB 70444
        [MUUO]                          ;112 DFMP 70450
        [MUUO]                          ;113 DFDV 70454
        [DFETCH AC&MEM] [DADD] [D TO AC.0]      ;114 DADD 70460
        [DFETCH AC&MEM] [DSUB] [D TO AC.0]      ;115 DSUB 70464
        [DFETCH AC&MEM] [DMUL] [Q TO AC.0]      ;116 DMUL 70470
        [DFETCH AC&MEM] [DDIV] [Q TO AC.0]      ;117 DDIV 70474
        [DFETCH MEM] [D TO AC]                  ;120 DMOVE 70500
        [DFETCH MEM] [DMOVNX] [D TO AC.0]       ;121 DMOVN 70504
        [FETCH MEM] [FIX] [IFETCH]              ;122 FIX 70510
.IF/FTEXTEND
        [EXTEND]                                ;123 EXTEND 70514
.ENDIF/FTEXTEND
.IFNOT/FTEXTEND
        [MUUO]                                  ;123 EXTEND 70514
.ENDIF/FTEXTEND
        [DFETCH AC] [D TO MEM]                  ;124 DMOVEM 70520
        [DFETCH AC] [DMOVNX] [D TO MEM]         ;125 DMOVNM 70524
        [FETCH MEM] [FIX] [FIXR]                ;126 FIXR 70530
        [FETCH MEM] [FLTR] [TO AC]              ;127 FLTR 70534
        [MUUO]                  ;130 UFA(obs) 70540
        [MUUO]                  ;131 DFN(obs) 70544
        [MUUO]                  ;132 FSC 70550
.IF/FTBYTE
        [FETCH AC&MEM] [ADJBP] [TO NOWHERE] [ADJBPO]    ;133 IBP ADJBP 70554
        [FETCH MEM(W)] [IP&S SETUP] [IP&S 5] [LDB]      ;134 ILDB 70560
```

```
        [FETCH MEM] [P&S SETUP] [MCE] [LDB]             ;135 LDB 70564
        [FETCH AC&MEM(W)] [IP&S SETUP] [IP&S 5] [DPB]   ;136 IDPB 70570
        [FETCH AC&MEM] [P&S SETUP] [MCE] [DPB]          ;137 DPB 70574
.ENDIF/FTBYTE
.IFNOT/FTBYTE
        [MUUO]                  ;133 IBP ADJBP 70554
        [MUUO]                  ;134 ILDB 70560
        [MUUO]                  ;135 LDB 70564
        [MUUO]                  ;136 IDPB 70570
        [MUUO]                  ;137 DPB 70574
.ENDIF/FTBYTE
        [FETCH AC&MEM] [FADX] [TO AC.0]         ;140 FAD 70600
        [MUUO]                                  ;141 FADL(obs) 70604
        [FETCH AC&MEM(W)] [FADX] [TO MEM]       ;142 FADM 70610
        [FETCH AC&MEM(W)] [FADX] [TO BOTH.0]    ;143 FADB 70614
        [FETCH AC&MEM] [FADRX] [TO AC.0]        ;144 FADR 70620
        [FETCH AC&I] [FADRX] [TO AC.0]          ;145 FADRI 70624
        [FETCH AC&MEM(W)] [FADRX] [TO MEM]      ;146 FADRM 70630
        [FETCH AC&MEM(W)] [FADRX] [TO BOTH.0]   ;147 FADRB 70634
        [FETCH AC&MEM] [FSBX] [TO AC.0]         ;150 FSB 70640
        [MUUO]                                  ;151 FSBL(obs) 70644
        [FETCH AC&MEM(W)] [FSBX] [TO MEM]       ;152 FSBM 70650
        [FETCH AC&MEM(W)] [FSBX] [TO BOTH.0]    ;153 FSBB 70654
        [FETCH AC&MEM] [FSBX] [TO AC.0]         ;154 FSBR 70660
        [FETCH AC&I] [FSBX] [TO AC.0]           ;155 FSBRI 70664
        [FETCH AC&MEM(W)] [FSBX] [TO MEM]       ;156 FSBRM 70670
        [FETCH AC&MEM(W)] [FSBX] [TO BOTH.0]    ;157 FSBRB 70674
        [FETCH AC&MEM] [FMPX] [TO AC.0]         ;160 FMP 70700
        [MUUO]                                  ;161 FMPL(obs) 70704
        [FETCH AC&MEM(W)] [FMPX] [TO MEM]       ;162 FMPM 70710
        [FETCH AC&MEM(W)] [FMPX] [TO BOTH.0]    ;163 FMPB 70714
        [FETCH AC&MEM] [FMPRX] [TO AC.0]        ;164 FMPR 70720
        [FETCH AC&MEM] [FMPRX] [TO AC.0]        ;165 FMPRI 70724
        [FETCH AC&MEM(W)] [FMPRX] [TO MEM]      ;166 FMPRM 70730
        [FETCH AC&MEM(W)] [FMPRX] [TO BOTH.0]   ;167 FMPRB 70734
        [FETCH AC&MEM] [FDVX] [TO AC.0]         ;170 FDV 70740
        [MUUO]                                  ;171 FDVL(obs) 70744
        [FETCH AC&MEM(W)] [FDVX] [TO MEM]       ;172 FDVM 70750
        [FETCH AC&MEM(W)] [FDVX] [TO BOTH.0]    ;173 FDVB 70754
        [FETCH AC&MEM] [FDVX] [TO AC.0]         ;174 FDVR 70760
        [FETCH AC&MEM] [FDVX] [TO AC.0]         ;175 FDVRI 70764
        [FETCH AC&MEM(W)] [FDVX] [TO MEM]       ;176 FDVRM 70770
        [FETCH AC&MEM(W)] [FDVX] [TO BOTH.0]    ;177 FDVRB 70774
        [FETCH MEM] [TO AC]                     ;200 MOVE 71000
.IF/FAST
        [MOVEI]                                 ;201 MOVEI 71004
.ENDIF/FAST
.IFNOT/FAST
        [FETCH I] [TO AC]                       ;201 MOVEI 71004
.ENDIF/FAST
        [FETCH AC] [TO MEM]                     ;202 MOVEM 71010
        [FETCH MEM(W)] [BACK TO SELF]           ;203 MOVES 71014
        [FETCH MEM] [MOVSX] [TO AC]             ;204 MOVS 71020
.IF/FAST
        [MOVSI]                                 ;205 MOVSI 71024
.ENDIF/FAST
.IFNOT/FAST
        [FETCH I] [MOVSX] [TO AC]               ;205 MOVSI 71024
.ENDIF/FAST
        [FETCH AC] [MOVSX] [TO MEM]             ;206 MOVSM 71030
```

```
        [FETCH MEM(W)] [MOVSX] [BACK TO SELF]   ;207 MOVSS 71034
        [FETCH MEM] [MOVNX] [TO AC]             ;210 MOVN 71040
        [FETCH I] [MOVNX] [TO AC]               ;211 MOVNI 71044
        [FETCH AC(MX)] [MOVNX] [TO MEM]         ;212 MOVNM 71050
        [FETCH MEM(W)] [MOVNX] [BACK TO SELF]   ;213 MOVNS 71054
        [FETCH MEM] [MOVMX] [TO AC]             ;214 MOVM 71060
.IF/FAST
        [MOVEI]                                 ;215 MOVMI (=MOVEI) 71064
.ENDIF/FAST
.IFNOT/FAST
        [FETCH I] [TO AC]                       ;215 MOVMI (=MOVEI) 71064
.ENDIF/FAST
        [FETCH AC(MX)] [MOVMX] [TO MEM]         ;216 MOVMM 71070
        [FETCH MEM(W)] [MOVMX] [BACK TO SELF]   ;217 MOVMS 71074
        [FETCH AC&MEM] [MULX] [IMULX] [TO AC.0]         ;220 IMUL 71100
        [FETCH AC&I] [MULX] [IMULX] [TO AC.0]           ;221 IMULI 71104
        [FETCH AC&MEM(W)] [MULX] [IMULX] [BACK TO MEM]  ;222 IMULM 71110
        [FETCH AC&MEM(W)] [MULX] [IMULX] [TO BOTH.0]    ;223 IMULB 71114
        [FETCH AC&MEM] [MULX] [D TO AC]         ;224 MUL 71120
        [FETCH AC&I] [MULX] [D TO AC]           ;225 MULI 71124
        [FETCH AC&MEM(W)] [MULX] [BACK TO MEM]  ;226 MULM 71130
        [FETCH AC&MEM(W)] [MULX] [TO MEM D AC]  ;227 MULB 71134
        [FETCH AC&MEM] [IDIVX] [D TO AC.0]      ;230 IDIV 71140
        [FETCH AC&I] [IDIVX] [D TO AC.0]        ;231 IDIVI 71144
        [FETCH AC&MEM(W)] [IDIVX] [BACK TO MEM] ;232 IDIVM 71150
        [FETCH AC&MEM(W)] [IDIVX] [TO MEM D AC] ;233 IDIVB 71154
        [FETCH AC&MEM] [DIVX] [D TO AC.0]       ;234 DIV 71160
        [FETCH AC&I] [DIVX] [D TO AC.0]         ;235 DIVI 71164
        [FETCH AC&MEM(W)] [DIVX] [BACK TO MEM]  ;236 DIVM 71170
        [FETCH AC&MEM(W)] [DIVX] [TO MEM D AC]  ;237 DIVB 71174
        [FETCH AC(MX)] [ASH SETUP] [ASH] [ASHR] ;240 ASH 71200
        [FETCH AC] [ASH SETUP] [ROT] [ROTR]     ;241 ROT 71204
        [FETCH AC] [ASH SETUP] [LSH] [LSHR]     ;242 LSH 71210
        [FETCH AC(MX)] [JFFO]                   ;243 JFFO 71214
        [DFETCH AC] [ASHC SETUP] [ASHC] [ASHCR] ;244 ASHC 71220
        [DFETCH AC] [LSHC SETUP] [ROTC] [ROTCR] ;245 ROTC 71224
        [DFETCH AC] [LSHC SETUP] [LSHC] [LSHCR] ;246 LSHC 71230
        [MUUO]                                  ;247 71234
        [FETCH AC&MEM(W)] [TO EACH]             ;250 EXCH 71240
.IF/FAST
        [FETCH AC] [BLT] [BLT+1N]               ,251 BLT 71244
.ENDIF/FAST
.IFNOT/FAST
        [FETCH AC] [BLT]                        ;251 BLT 71244
.ENDIF/FAST
        [FETCH AC] [AOBJX] [JXGE] [TO AC.0]     ;252 AOBJP 71250
        [FETCH AC] [AOBJX] [JXL] [TO AC.0]      ;253 AOBJN 71254
        [JRST] [MCE] [MCE] [IFETCH]             ;254 JRST 71260
        [JFCL] [MCE] [MCE] [IFETCH]             ;255 JFCL 71264
        [FETCH MEM] [XCT]                       ;256 XCT 71270
        [MAP]                                   ;257 MAP 71274
        [FETCH AC] [CHK PC SECT] [PUSHJ]        ;260 PUSHJ 71300
        [FETCH AC&MEM] [PUSH]                   ;261 PUSH 71304
        [FETCH AC] [CHK PC SECT] [POP]          ;262 POP 71310
        [FETCH AC] [CHK PC SECT] [POPJ]         ;263 POPJ 71314
        [CHK PC SECT] [JSR] [JSR SETUP]         ;264 JSR 71320
        [CHK PC SECT] [JSP] [JSR SETUP]         ;265 JSP 71324
        [FETCH AC] [JSA]                        ;266 JSA 71330
        [FETCH AC] [JRA]                        ;267 JRA 71334
        [FETCH AC&MEM] [ADDX] [TO AC]           ;270 ADD 71340
```

```
[FETCH AC&I] [ADDX] [TO AC]                ;271 ADDI 71344
[FETCH AC&MEM(W)] [ADDX] [BACK TO MEM]     ;272 ADDM 71350
[FETCH AC&MEM(W)] [ADDX] [TO BOTH.O]       ;273 ADDB 71354
[FETCH AC&MEM] [SUBX] [TO AC]              ;274 SUB 71360
[FETCH AC&I] [SUBX] [TO AC]                ;275 SUBI 71364
[FETCH AC&MEM(W)] [SUBX] [BACK TO MEM]     ;276 SUBM 71370
[FETCH AC&MEM(W)] [SUBX] [TO BOTH.O]       ;277 SUBB 71374
[TO NOWHERE]                               ;300 CAI 71400
[FETCH AC&I] [CAX] [SXL] [IFETCH]##[JXA]        ;301 CAIL 71404 JRST 1,
[FETCH AC&I] [CAX] [SXE] [IFETCH]##[JRSTF]      ;302 CAIE 71410 JRST 2,
[FETCH AC&I] [CAX] [SXLE] [IFETCH]##[MUUO]      ;303 CAILE 71414 JRST 3,
[FETCH AC&I] [CAX] [SXA] [IFETCH]##[HALT]       ;304 CAIA 71420 JRST 4,
[FETCH AC&I] [CAX] [SXGE] [IFETCH]##[XJRSTF]    ;305 CAIGE 71424 JRST 5,
[FETCH AC&I] [CAX] [SXN] [IFETCH]##[XJEN]       ;306 CAIN 71430 JRST 6,
[FETCH AC&I] [CAX] [SXG] [IFETCH]##[XPCW]       ;307 CAIG 71434 JRST 7,
[FETCH MEM] [IFETCH]##[JRST 10]                 ;310 CAM 71440 JRST 10,
[FETCH AC&MEM] [CAX] [SXL] [IFETCH]##[MUUO]     ;311 CAML 71444 JRST 11,
[FETCH AC&MEM] [CAX] [SXE] [IFETCH]##[JEN]      ;312 CAME 71450 JRST 12,
[FETCH AC&MEM] [CAX] [SXLE] [IFETCH]##[MUUO]    ;313 CAMLE 71454 JRST 13,
[FETCH AC&MEM] [CAX] [SXA] [IFETCH]##[SFM]      ;314 CAMA 71460 JRST 14,
[FETCH AC&MEM] [CAX] [SXGE] [IFETCH]##[MUUO]    ;315 CAMGE 71464 JRST 15,
[FETCH AC&MEM] [CAX] [SXN] [IFETCH]##[MUUO]     ;316 CAMN 71470 JRST 16,
[FETCH AC&MEM] [CAX] [SXG] [IFETCH]##[MUUO]     ;317 CAMG 71474 JRST 17,
[TO NOWHERE]                               ;320 JUMP 71500
[FETCH AC(MX)] [JXL] [MCE] [IFETCH]        ;321 JUMPL 71504
[FETCH AC(MX)] [JXE] [MCE] [IFETCH]        ;322 JUMPE 71510
[FETCH AC(MX)] [JXLE] [MCE] [IFETCH]       ;323 JUMPLE 71514
[JXA] [MCE] [MCE] [IFETCH]                 ;324 JUMPA 71520
[FETCH AC(MX)] [JXGE] [MCE] [IFETCH]       ;325 JUMPGE 71524
[FETCH AC(MX)] [JXN] [MCE] [IFETCH]        ;326 JUMPN 71530
[FETCH AC(MX)] [JXG] [MCE] [IFETCH]        ;327 JUMPG 71534
[FETCH MEM] [SKIPX]                        ;330 SKIP 71540
[FETCH MEM] [SXL] [MCE] [SKIPX]            ;331 SKIPL 71544
[FETCH MEM] [SXE] [MCE] [SKIPX]            ;332 SKIPE 71550
[FETCH MEM] [SXLE] [MCE] [SKIPX]           ;333 SKIPLE 71554
[FETCH MEM] [SXA] [MCE] [SKIPX]            ;334 SKIPA 71560
[FETCH MEM] [SXGE] [MCE] [SKIPX]           ;335 SKIPGE 71564
[FETCH MEM] [SXN] [MCE] [SKIPX]            ;336 SKIPN 71570
[FETCH MEM] [SXG] [MCE] [SKIPX]            ;337 SKIPG 71574
[FETCH AC] [AOXX] [TO AC]                  ;340 AOJ 71600
[FETCH AC] [AOXX] [JXL] [TO AC]            ;341 AOJL 71604
[FETCH AC] [AOXX] [JXE] [TO AC]            ;342 AOJE 71610
[FETCH AC] [AOXX] [JXLE] [TO AC]           ;343 AOJLE 71614
[FETCH AC] [AOXX] [JXA] [TO AC]            ;344 AOJA 71620
[FETCH AC] [AOXX] [JXGE] [TO AC]           ;345 AOJGE 71624
[FETCH AC] [AOXX] [JXN] [TO AC]            ;346 AOJN 71630
[FETCH AC] [AOXX] [JXG] [TO AC]            ;347 AOJG 71634
[FETCH MEM(W)] [AOXX] [BACK TO SELF]            ;350 AOS 71640
[FETCH MEM(W)] [AOXX] [SXL] [BACK TO SELF]      ;351 AOSL 71644
[FETCH MEM(W)] [AOXX] [SXE] [BACK TO SELF]      ;352 AOSE 71650
[FETCH MEM(W)] [AOXX] [SXLE] [BACK TO SELF]     ;353 AOSLE 71654
[FETCH MEM(W)] [AOXX] [SXA] [BACK TO SELF]      ;354 AOSA 71660
[FETCH MEM(W)] [AOXX] [SXGE] [BACK TO SELF]     ;355 AOSGE 71664
[FETCH MEM(W)] [AOXX] [SXN] [BACK TO SELF]      ;356 AOSN 71670
[FETCH MEM(W)] [AOXX] [SXG] [BACK TO SELF]      ;357 AOSG 71674
[FETCH AC] [SOXX] [TO AC]                  ;360 SOJ 71700
[FETCH AC] [SOXX] [JXL] [TO AC]            ;361 SOJL 71704
[FETCH AC] [SOXX] [JXE] [TO AC]            ;362 SOJE 71710
[FETCH AC] [SOXX] [JXLE] [TO AC]           ;363 SOJLE 71714
[FETCH AC] [SOXX] [JXA] [TO AC]            ;364 SOJA 71720
```

```
[FETCH AC] [SOXX] [JXGE] [TO AC]          ;365 SOJGE 71724
[FETCH AC] [SOXX] [JXN] [TO AC]           ;366 SOJN 71730
[FETCH AC] [SOXX] [JXG] [TO AC]           ;367 SOJG 71734
[FETCH MEM(W)] [SOXX] [BACK TO SELF]              ;370 SOS 71740
[FETCH MEM(W)] [SOXX] [SXL] [BACK TO SELF]        ;371 SOSL 71744
[FETCH MEM(W)] [SOXX] [SXE] [BACK TO SELF]        ;372 SOSE 71750
[FETCH MEM(W)] [SOXX] [SXLE] [BACK TO SELF]       ;373 SOSLE 71754
[FETCH MEM(W)] [SOXX] [SXA] [BACK TO SELF]        ;374 SOSA 71760
[FETCH MEM(W)] [SOXX] [SXGE] [BACK TO SELF]       ;375 SOSGE 71764
[FETCH MEM(W)] [SOXX] [SXN] [BACK TO SELF]        ;376 SOSN 71770
[FETCH MEM(W)] [SOXX] [SXG] [BACK TO SELF]        ;377 SOSG 71774
[SETZX] [TO AC]                           ;400 SETZ 72000
[SETZX] [TO AC]                           ;401 SETZI 72004
[SETZX] [TO MEM]                          ;402 SETZM 72010
[SETZX] [TO BOTH]                         ;403 SETZB 72014
[FETCH AC&MEM] [ANDX] [TO AC]             ;404 AND 72020
[FETCH AC&I] [ANDX] [TO AC]               ;405 ANDI 72024
[FETCH AC&MEM(W)] [ANDX] [BACK TO MEM]    ;406 ANDM 72030
[FETCH AC&MEM(W)] [ANDX] [TO BOTH]        ;407 ANDB 72034
[FETCH AC&MEM] [ANDCAX] [TO AC]           ;410 ANDCA 72040
[FETCH AC&I] [ANDCAX] [TO AC]             ;411 ANDCAI 72044
[FETCH AC&MEM(W)] [ANDCAX] [BACK TO MEM]          ;412 ANDCAM 72050
[FETCH AC&MEM(W)] [ANDCAX] [TO BOTH]      ;413 ANDCAB 72054
[FETCH MEM] [TO AC]                       ;414 SETM 72060
[XMOVEI] [MCE] [TO AC]                    ;415 SETMI XMOVEI 72064
[FETCH MEM(W)] [BACK TO MEM]              ;416 SETMM 72070
[FETCH MEM(W)] [TO BOTH]                  ;417 SETMB 72074
[FETCH AC&MEM] [ANDCMX] [TO AC]           ;420 ANDCM 72100
[FETCH AC&I] [ANDCMX] [TO AC]             ;421 ANDCMI 72104
[FETCH AC&MEM(W)] [ANDCMX] [BACK TO MEM]          ;422 ANDCMM 72110
[FETCH AC&MEM(W)] [ANDCMX] [TO BOTH]      ;423 ANDCMB 72114
[TO NOWHERE]                              ;424 SETA (no-op) 72120
[TO NOWHERE]                              ;425 SETAI (no-op) 72124
[FETCH AC] [TO MEM]                       ;426 SETAM (= MOVEM) 72130
[FETCH AC] [TO MEM]                       ;427 SETAB (= MOVEM) 72134
[FETCH AC&MEM] [XORX] [TO AC]             ;430 XOR 72140
[FETCH AC&I] [XORX] [TO AC]               ;431 XORI 72144
[FETCH AC&MEM(W)] [XORX] [BACK TO MEM]    ;432 XORM 72150
[FETCH AC&MEM(W)] [XORX] [TO BOTH]        ;433 XORB 72154
[FETCH AC&MEM] [IORX] [TO AC]             ;434 IOR 72160
[FETCH AC&I] [IORX] [TO AC]               ;435 IORI 72164
[FETCH AC&MEM(W)] [IORX] [BACK TO MEM]    ;436 IORM 72170
[FETCH AC&MEM(W)] [IORX] [TO BOTH]        ;437 IORB 72174
[FETCH AC&MEM] [ANDCBX] [TO AC]           ;440 ANDCB 72200
[FETCH AC&I] [ANDCBX] [TO AC]             ;441 ANDCBI 72204
[FETCH AC&MEM(W)] [ANDCBX] [BACK TO MEM]          ;442 ANDCBM 72210
[FETCH AC&MEM(W)] [ANDCBX] [TO BOTH]      ;443 ANDCBB 72214
[FETCH AC&MEM] [EQVX] [TO AC]             ;444 EQV 72220
[FETCH AC&I] [EQVX] [TO AC]               ;445 EQVI 72224
[FETCH AC&MEM(W)] [EQVX] [BACK TO MEM]    ;446 EQVM 72230
[FETCH AC&MEM(W)] [EQVX] [TO BOTH]        ;447 EQVB 72234
[FETCH AC] [SETCAX] [TO AC]               ;450 SETCA 72240
[FETCH AC] [SETCAX] [TO AC]               ;451 SETCAI 72244
[FETCH AC] [SETCAX] [TO MEM]              ;452 SETCAM 72250
[FETCH AC] [SETCAX] [TO BOTH]             ;453 SETCAB 72254
[FETCH AC&MEM] [ORCAX] [TO AC]            ;454 ORCA 72260
[FETCH AC&I] [ORCAX] [TO AC]              ;455 ORCAI 72264
[FETCH AC&MEM(W)] [ORCAX] [BACK TO MEM]   ;456 ORCAM 72270
[FETCH AC&MEM(W)] [ORCAX] [TO BOTH]       ;457 ORCAB 72274
[FETCH MEM] [SETCMX] [TO AC]              ;460 SETCM 72300
```

```
        [FETCH I] [SETCMX] [TO AC]                ;461 SETCMI 72304
        [FETCH MEM(W)] [SETCMX] [BACK TO MEM]     ;462 SETCMM 72310
        [FETCH MEM(W)] [SETCMX] [TO BOTH]         ;463 SETCMB 72314
        [FETCH AC&MEM] [ORCMX] [TO AC]            ;464 ORCM 72320
        [FETCH AC&I] [ORCMX] [TO AC]              ;465 ORCMI 72324
        [FETCH AC&MEM(W)] [ORCMX] [BACK TO MEM]   ;466 ORCMM 72330
        [FETCH AC&MEM(W)] [ORCMX] [TO BOTH]       ;467 ORCMB 72334
        [FETCH AC&MEM] [ORCBX] [TO AC]            ;470 ORCB 72340
        [FETCH AC&I] [ORCBX] [TO AC]              ;471 ORCBI 72344
        [FETCH AC&MEM(W)] [ORCBX] [BACK TO MEM]   ;472 ORCBM 72350
        [FETCH AC&MEM(W)] [ORCBX] [TO BOTH]       ;473 ORCBB 72354
        [SETOX] [TO AC]                           ;474 SETO 72360
        [SETOX] [TO AC]                           ;475 SETOI 72364
        [SETOX] [TO MEM]                          ;476 SETOM 72370
        [SETOX] [TO BOTH]                         ;477 SETOB 72374
        [FETCH AC&MEM] [HLL] [TO AC]              ;500 HLL 72400
        [FETCH AC] [XHLLI] [TO AC.0]              ;501 HLLI XHLLI 72404
        [FETCH AC&MEM(W)] [HLLM] [TO MEM]         ;502 HLLM 72410
        [FETCH MEM(W)] [BACK TO SELF]             ;503 HLLS 72414
        [FETCH AC&MEM] [HRL] [AC TO AC]           ;504 HRL 72420
        [FETCH AC&I] [HRL] [AC TO AC]             ;505 HRLI 72424
        [FETCH AC&MEM(W)] [HRLM] [TO MEM]         ;506 HRLM 72430
        [FETCH AC&MEM(W)] [HRLS] [BACK TO SELF]   ;507 HRLS 72434
        [FETCH MEM] [HLLZX] [MCE] [TO AC]         ;510 HLLZ 72440
        [FETCH I] [HLLZX] [MCE] [TO AC]           ;511 HLLZI 72444
        [FETCH AC] [HLLZX] [MCE] [TO MEM]         ;512 HLLZM 72450
        [FETCH MEM(W)] [HLLZX] [MCE] [BACK TO SELF]     ;513 HLLZS 72454
        [FETCH MEM] [MOVSX] [HLLZX] [TO AC]       ;514 HRLZ 72460
.IF/FAST
        [MOVSI]                                   ;515 HRLZI 72464
.ENDIF/FAST
.IFNOT/FAST
        [FETCH I] [MOVSX] [HLLZX] [TO AC]         ;515 HRLZI 72464
.ENDIF/FAST
        [FETCH AC] [MOVSX] [HLLZX] [TO MEM]       ;516 HRLZM 72470
        [FETCH MEM(W)] [MOVSX] [HLLZX] [BACK TO SELF]   ;517 HRLZS 72474
        [FETCH MEM] [HLLOX] [MCE] [TO AC]         ;520 HLLO 72500
        [FETCH I] [HLLOX] [MCE] [TO AC]           ;521 HLLOI 72504
        [FETCH AC] [HLLOX] [MCE] [TO MEM]         ;522 HLLOM 72510
        [FETCH MEM(W)] [HLLOX] [MCE] [BACK TO SELF]     ;523 HLLOS 72514
        [FETCH MEM] [MOVSX] [HLLOX] [TO AC]             ;524 HRLO 72520
        [FETCH I] [MOVSX] [HLLOX] [TO AC]               ;525 HRLOI 72524
        [FETCH AC] [MOVSX] [HLLOX] [TO MEM]             ;526 HRLOM 72530
        [FETCH MEM(W)] [MOVSX] [HLLOX] [BACK TO SELF]   ;527 HRLOS 72534
        [FETCH MEM] [HLLEX] [MCE] [TO AC]               ;530 HLLE 72540
        [FETCH I] [HLLEX] [MCE] [TO AC]           ;531 HLLEI 72544
        [FETCH AC(MX)] [HLLEX] [MCE] [TO MEM]           ;532 HLLEM 72550
        [FETCH MEM(W)] [HLLEX] [MCE] [BACK TO SELF]     ;533 HLLES 72554
        [FETCH MEM] [MOVSX] [HRLEX] [TO AC]             ;534 HRLE 72560
        [FETCH I] [MOVSX] [HRLEX] [TO AC]               ;535 HRLEI 72564
        [FETCH AC(MX)] [MOVSX] [HRLEX] [TO MEM]         ;536 HRLEM 72570
        [FETCH MEM(W)] [MOVSX] [HRLEX] [BACK TO SELF]   ;537 HRLES 72574
        [FETCH AC&MEM] [HRR] [TO AC]              ;540 HRR 72600
        [FETCH AC&I] [HRR] [TO AC]                ;541 HRRI 72604
        [FETCH AC&MEM(W)] [HRRM] [TO MEM]         ;542 HRRM 72610
        [FETCH MEM(W)] [BACK TO SELF]             ;543 HRRS 72614
        [FETCH AC&MEM] [HLR] [AC TO AC]           ;544 HLR 72620
        [FETCH AC&I] [HLR] [AC TO AC]             ;545 HLRI 72624
        [FETCH AC&MEM(W)] [HLRM] [TO MEM]         ;546 HLRM 72630
        [FETCH AC&MEM(W)] [HLRS] [BACK TO SELF]   ;547 HLRS 72634
```

```
        [FETCH MEM] [HRRZX] [MCE] [TO AC]               ;550 HRRZ 72640
.IF/FAST
        [MOVEI]                                         ;551 HRRZI 72644
.ENDIF/FAST
 IFNOT/FAST
        [FETCH I] [HRRZX] [MCE] [TO AC]                 ;551 HRRZI 72644
.ENDIF/FAST
        [FETCH AC] [HRRZX] [MCE] [TO MEM]                       ;552 HRRZM 72650
        [FETCH MEM(W)] [HRRZX] [MCE] [BACK TO SELF]             ;553 HRRZS 72654
        [FETCH MEM] [MOVSX] [HRRZX] [TO AC]                     ;554 HLRZ 72660
        [FETCH I] [MOVSX] [HRRZX] [TO AC]                       ;555 HLRZI 72664
        [FETCH AC] [MOVSX] [HRRZX] [TO MEM]                     ;556 HLRZM 72670
        [FETCH MEM(W)] [MOVSX] [HRRZX] [BACK TO SELF]           ;557 HLRZS 72674
        [FETCH MEM] [HRROX] [MCE] [TO AC]                       ;560 HRRO 72700
        [FETCH I] [HRROX] [MCE] [TO AC]                 ;561 HRROI 72704
        [FETCH AC] [HRROX] [MCE] [TO MEM]                       ;562 HRROM 72710
        [FETCH MEM(W)] [HRROX] [MCE] [BACK TO SELF]             ;563 HRROS 72714
        [FETCH MEM] [MOVSX] [HRROX] [TO AC]                     ;564 HLRO 72720
        [FETCH I] [MOVSX] [HRROX] [TO AC]                       ;565 HLROI 72724
        [FETCH AC] [MOVSX] [HRROX] [TO MEM]                     ;566 HLROM 72730
        [FETCH MEM(W)] [MOVSX] [HRROX] [BACK TO SELF]           ;567 HLROS 72734
        [FETCH MEM] [HRREX] [MCE] [TO AC]                       ;570 HRRE 72740
        [FETCH I] [HRREX] [MCE] [TO AC]                 ;571 HRREI 72744
        [FETCH AC(MX)] [HRREX] [MCE] [TO MEM]                   ;572 HRREM 72750
        [FETCH MEM(W)] [HRREX] [MCE] [BACK TO SELF]             ;573 HRRES 72754
        [FETCH MEM] [MOVSX] [HRREX] [TO AC]                     ;574 HLRE 72760
        [FETCH I] [MOVSX] [HRREX] [TO AC]                       ;575 HLREI 72764
        [FETCH AC(MX)] [MOVSX] [HRREX] [TO MEM]                 ;576 HLREM 72770
        [FETCH MEM(W)] [MOVSX] [HRREX] [BACK TO SELF]           ;577 HLRES 72774
        [TO NOWHERE]                    ;600 TRN 73000
        [TO NOWHERE]                    ;601 TLN 73004
        [TRX] [TXE] [IFETCH]            ;602 TRNE 73010
        [TLX] [TXE] [IFETCH]            ;603 TLNE 73014
        [TRX] [TXA] [IFETCH]            ;604 TRNA 73020
        [TLX] [TXA] [IFETCH]            ;605 TLNA 73024
        [TRX] [TXN] [IFETCH]            ;606 TRNN 73030
        [TLX] [TXN] [IFETCH]            ;607 TLNN 73034
        [TDX] [IFETCH]                  ;610 TDN 73040
        [TSX] [IFETCH]                  ;611 TSN 73044
        [TDX] [TXE] [IFETCH]            ;612 TDNE 73050
        [TSX] [TXE] [IFETCH]            ;613 TSNE 73054
        [TDX] [TXA] [IFETCH]            ;614 TDNA 73060
        [TSX] [TXA] [IFETCH]            ;615 TSNA 73064
        [TDX] [TXN] [IFETCH]            ;616 TDNN 73070
        [TSX] [TXN] [IFETCH]            ;617 TSNN 73074
        [TRX] [TXZ]                     ;620 TRZ 73100
        [TLX] [TXZ]                     ;621 TLZ 73104
        [TRX] [TXE] [TXZ]               ;622 TRZE 73110
        [TLX] [TXE] [TXZ]               ;623 TLZE 73114
        [TRX] [TXA] [TXZ]               ;624 TRZA 73120
        [TLX] [TXA] [TXZ]               ;625 TLZA 73124
        [TRX] [TXN] [TXZ]               ;626 TRZN 73130
        [TLX] [TXN] [TXZ]               ;627 TLZN 73134
        [TDX] [TXZ]                     ;630 TDZ 73140
        [TSX] [TXZ]                     ;631 TSZ 73144
        [TDX] [TXE] [TXZ]               ;632 TDZE 73150
        [TSX] [TXE] [TXZ]               ;633 TSZE 73154
        [TDX] [TXA] [TXZ]               ;634 TDZA 73160
        [TSX] [TXA] [TXZ]               ;635 TSZA 73164
        [TDX] [TXN] [TXZ]               ;636 TDZN 73170
```

```
        [TSX] [TXN] [TXZ]               ;637 TSZN 73174
        [TRX] [TXC]                     ;640 TRC 73200
        [TLX] [TXC]                     ;641 TLC 73204
        [TRX] [TXE] [TXC]               ;642 TRCE 73210
        [TLX] [TXE] [TXC]               ;643 TLCE 73214
        [TRX] [TXA] [TXC]               ;644 TRCA 73220
        [TLX] [TXA] [TXC]               ;645 TLCA 73224
        [TRX] [TXN] [TXC]               ;646 TRCN 73230
        [TLX] [TXN] [TXC]               ;647 TLCN 73234
        [TDX] [TXC]                     ;650 TDC 73240
        [TSX] [TXC]                     ;651 TSC 73244
        [TDX] [TXE] [TXC]               ;652 TDCE 73250
        [TSX] [TXE] [TXC]               ;653 TSCE 73254
        [TDX] [TXA] [TXC]               ;654 TDCA 73260
        [TSX] [TXA] [TXC]               ;655 TSCA 73264
        [TDX] [TXN] [TXC]               ;656 TDCN 73270
        [TSX] [TXN] [TXC]               ;657 TSCN 73274
        [TRX] [TXO]                     ;660 TRO 73300
        [TLX] [TXO]                     ;661 TLO 73304
        [TRX] [TXE] [TXO]               ;662 TROE 73310
        [TLX] [TXE] [TXO]               ;663 TLOE 73314
        [TRX] [TXA] [TXO]               ;664 TROA 73320
        [TLX] [TXA] [TXO]               ;665 TLOA 73324
        [TRX] [TXN] [TXO]               ;666 TRON 73330
        [TLX] [TXN] [TXO]               ;667 TLON 73334
        [TDX] [TXO]                     ;670 TDO 73340
        [TSX] [TXO]                     ;671 TSO 73344
        [TDX] [TXE] [TXO]               ;672 TDOE 73350
        [TSX] [TXE] [TXO]               ;673 TSOE 73354
        [TDX] [TXA] [TXO]               ;674 TDOA 73360
        [TSX] [TXA] [TXO]               ;675 TSOA 73364
        [TDX] [TXN] [TXO]               ;676 TDON 73370
        [TSX] [TXN] [TXO]               ;677 TSON 73374
        [IO]                    ;700 IO 73400
        [IO]                    ;701 IO 73404
        [IO]                    ;702 IO 73410
        [IO]                    ;703 IO 73414
        [IO]                    ;704 IO 73420
        [IO]                    ;705 IO 73424
        [IO]                    ;706 IO 73430
        [IO]                    ;707 IO 73434
        [IO EFA] [FETCH AC&IO] [TIOE]   ;710 TIOE 73440
        [IO EFA] [FETCH AC&IO] [TION]   ;711 TION 73444
        [IO EFA] [FETCH IO] [TO AC]     ;712 RDIO 73450
        [IO EFA] [WRIO]                 ;713 WRIO 73454
        [IO EFA] [FETCH AC&IO] [BSIO]   ;714 BSIO 73460
        [IO EFA] [FETCH AC&IO] [BCIO]   ;715 BCIO 73464
        [IO]                            ;716 IO 73470
        [IO]                            ;717 IO 73474
        [MUUO]                  ;720 TIOEB 73500
        [MUUO]                  ;721 TIONB 73504
.IF/DEBUGTTY
        [RDTTY]                 ;722 RDIOB 73510
.ENDIF/DEBUGTTY
.IFNOT/DEBUGTTY
        [MUUO]                  ;722 RDIOB 73510
.ENDIF/DEBUGTTY
        [MUUO]                  ;723 WRIOB 73514
        [MUUO]                  ;724 BSIOB 73520
        [MUUO]                  ;725 BCIOB 73524
```

```
        [IO]                    ;726 IO 73530
        [IO]                    ;727 IO 73534
        [IO]                    ;730 IO 73540
        [IO]                    ;731 IO 73544
        [IO]                    ;732 IO 73550
        [IO]                    ;733 IO 73554
        [IO]                    ;734 IO 73560
        [IO]                    ;735 IO 73564
        [IO]                    ;736 IO 73570
        [IO]                    ;737 IO 73574
.IF/FTCIS
        [CIS SETUP] [MOVC]      ;740 IO 73740
        [CIS SETUP] [CMPC]      ;741 IO 73744
        [CIS SETUP] [MOVCV]     ;742 IO 73610
        [CIS SETUP] [CMPCV]     ;743 IO 73614
        [CIS SETUP] [CMPND]     ;744 IO 73620
        [CIS SETUP] [ADDND]     ;745 IO 73624
        [CIS SETUP] [SUBND]     ;746 IO 73630
        [CIS SETUP] [MOVND]     ;747 IO 73634
        [CIS SETUP] [CVTNDB]    ;750 IO 73640
        [CIS SETUP] [CMPP]      ;751 IO 73644
        [CIS SETUP] [ADDP]      ;752 IO 73650
        [CIS SETUP] [SUBP]      ;753 IO 73654
        [CIS SETUP] [MOVP]      ;754 IO 73660
        [CIS SETUP] [CHOP]      ;755 IO 73664
.ENDIF/FTCIS
.IFNOT/FTCIS
        [IO 4]                  ;740 IO 73600
        [IO 4]                  ;741 IO 73604
        [IO 4]                  ;742 IO 73610
        [IO 4]                  ;743 IO 73614
        [IO 4]                  ;744 IO 73620
        [IO 4]                  ;745 IO 73624
        [IO 4]                  ;746 IO 73630
        [IO 4]                  ;747 IO 73634
        [IO 4]                  ;750 IO 73640
        [IO 4]                  ;751 IO 73644
        [IO 4]                  ;752 IO 73650
        [IO 4]                  ;753 IO 73654
        [IO 4]                  ;754 IO 73660
        [IO 4]                  ;755 IO 73664
.ENDIF/FTCIS
        [IO 4]                  ;756 IO 73670
        [IO 4]                  ;757 IO 73674
        [IO 4]                  ;760 IO 73700
        [IO 4]                  ;761 IO 73704
        [IO 4]                  ;762 IO 73710
        [IO 4]                  ;763 IO 73714
        [IO 4]                  ;764 IO 73720
        [IO 4]                  ;765 IO 73724
        [IO 4]                  ;766 IO 73730
        [IO 4]                  ;767 IO 73734
        [IO 4]                  ;770 IO 73740
        [IO 4]                  ;771 IO 73744
        [IO 4]                  ;772 IO 73750
        [IO 4]                  ;773 IO 73754
        [IO 4]                  ;774 IO 73760
        [IO 4]                  ;775 IO 73764
        [IO 4]                  ;776 IO 73770
        [IO 4]                  ;777 IO 73774
```