; TABLE OF CONTENTS

```
; 1     ;<MEIX.REX>REX05.MIC.3, 27-Jul-84 19:06:27, Edit by LOUGHEED
; 2     ; Implement timeout of Massbus transfers.  Set Q register to zero at the start
; 3     ;  of a transfer (when GO is asserted) and increment it every pass through
; 4     ;  the IDLE loop.  If the Q register becomes negative, set bit 11 in the error
; 5     ;  register and shutdown the transfer.
; 6     ; Clean up and speed up of control function dispatch table
; 7     ;<MEIS.REX>REX04.MIC.2, 12-Jul-84 20:55:16, Edit by LOUGHEED
; 8     ; Control Function microcode no longer checks for TX or RX available
; 9     ;  before setting GO.  This allows two data boards on the same controller.
; 10    ; Bump contents of version register - we are now REX version 1003
; 11
```

```
; 12
; 13    .TITLE "MEIS Control Microcode - Mach II"
; 14
; 15    ;Write PROM files using WREX2.SAI
; 16
; 17    ;MICROCODE VERSION 001003
; 18
; 19    ;Comments and reminders
; 20
; 21    ;Register 22 is now the Microcode Version Register.
; 22    ;The high byte is the major rev. and the low byte is the minor rev.
; 23
; 24    ;Writing to the Receiver Buffer Absolute Address Register, register 57, causes
; 25    ;the Rbuf Pointer, which is internal to the Data Board, to be loaded into
; 26    ;the Diagnostic Register, register 56.
; 27
; 28    ;19-AUG-82 Added jump field specifications (J/) to field select branches.
; 29    ;       (e.g. COND/REG_SEL, J/R.FIELD)
; 30
; 31    ;15-SEP-82 Fixed Absolute Address Register write. It now primes the register
; 32    ;for a subsequent read.
; 33    ;The Diagnostic register is now the Q-register in the 2901's.
; 34    ;Fixed some comma problems (added a few that were missing, deleted a few extra)
; 35    ;Reversed a couple of COND/ZERO tests that were backwards.
; 36
; 37    ;24-NOV-82 Created REX03. Changed PEXIT to return to IDLE via a jump, rather
; 38    ;than a COND/DEMAND. Changed all Control Function code to return via a
; 39    ;COND/DEMAND, J/IDLE. (The beginning of a Control Register write insures that
; 40    ;TRA is set long before the Control Function return.)
; 41
; 42
```

```
; 43
; 44    ; EXPLANATION OF NEXT ADDRESS LOGIC
; 45
; 46    ;       If (TEST SELECT GO .AND. GO) then RSEQR ADRS is from RMR field,
; 47    ;       else RSEQR ADRS is from field selected by RCON TEST SEL 5 and 6:
; 48    ;
; 49    ;               6 5     RSEQR ADRS field
; 50    ;               ---     ----------------
; 51    ;               0 0     JUMP ADRS
; 52    ;               0 1     MASS REG SEL
; 53    ;               1 0     RBUS (for Function Select)
; 54    ;               1 1     RMR
; 55    ;
; 56    ;       The least significant 2 bits of the above fields are selected by
; 57    ;       RCON TEST SELECT 1-4:
; 58    ;
; 59    ;               4321    RSEQR ADRS field, bits 1 and 0
; 60    ;               ----    ------------------------------
; 61    ;               0000    JUMP ADRS 1     JUMP ADRS 0
; 62    ;               0001    MASS REG SEL 1  MASS REG SEL 0
; 63    ;               0010    RBUS 1          RBUS 0
; 64    ;               0011    RMR 1           RMR 0
; 65    ;               0100    CPA             GO
; 66    ;               0101    ACQ 1           ACQ 0
; 67    ;               0110    EXC             EOP
; 68    ;               0111    END EBL         DEM
; 69    ;
; 70    ;               1000    JUMP ADRS 1     ZERO
; 71    ;               1001    JUMP ADRS 1     NEG
; 72    ;               1010    JUMP ADRS 1     XPORT REPLY
; 73    ;               1011    JUMP ADRS 1     DPA
; 74    ;               1100    JUMP ADRS 1     ADD TRAILER L
; 75    ;               1101    JUMP ADRS 1     MBC WRITE MODE
; 76    ;               1110    JUMP ADRS 1     SELECT BOARD 1
; 77    ;               1111    JUMP ADRS 1     CPA
; 78
; 79
```

```
; 80    .TOC "Field Definitions"
; 81
; 82    ;Pipeline next-address field
; 83
; 84    ;The Jump field is now 9 bits (512 words)
; 85    J/=0,9,8,+
; 86
; 87    ;Condition code selection
; 88
; 89    COND/=0,6,15,D
; 90            JUMP_ADRS       =00     ;default field select
; 91            REG_SEL         =21     ;needs J/ field
; 92            FUNC_SEL        =42     ;needs J/ field
; 93            RMR             =60     ;selects the RMR field
; 94
; 95            spare03         =03
; 96
; 97            CPA             =04
; 98            DPA             =05
; 99            NEG             =06
; 100           ZERO            =07
; 101
; 102           XPORT           =10
; 103           DEMAND          =11
; 104           ADD_TRAILER_L   =12
; 105           SEL_BOARD_1     =13
; 106
; 107           GO.MBC_WRITE    =14
; 108           ACQ1.ACQ0       =15
; 109           EXC.EOP         =16
; 110           END_EBL.DEM     =17
; 111
; 112
```

```
; 113   ; Register allocation in 2901 ALU
; 114
; 115   ALUA/=17,4,47,D
; 116           control.reg     =00     ;MEIS control register
; 117           error.reg       =01     ;MEIS error register
; 118           ether.adrs      =02     ;MEIS ethernet address register
; 119           abs.adrs        =03     ;MEIS receiver buffer absolute address register
; 120
; 121           rbuf.adrs       =04     ;MEIS receiver buffer address register
; 122           rbuf.data       =05     ;MEIS receiver buffer data register
; 123           rbuf.status     =06     ;MEIS receiver buffer status register
; 124           version         =07     ;MEIS microcode version register
; 125
; 126           header          =10     ;MEIS packet header count register
; 127           trailer         =11     ;MEIS packet trailer register, transmit
; 128           slave.id        =12     ;MEIS slave id register
; 129           old.status.reg  =13     ;previous status register (for ATA detect)
; 130
; 131           bit00           =14     ;bit 00 = 1
; 132           bit03           =15     ;bit 03 = 1
; 133           bit12           =16     ;bit 12 = 1
; 134           temp            =17     ;temporary holding register
; 135
; 136   ALUB/=17,4,43,D
; 137           control.reg     =00     ;MEIS control register
; 138           error.reg       =01     ;MEIS error register
; 139           ether.adrs      =02     ;MEIS ethernet address register
; 140           abs.adrs        =03     ;MEIS receiver buffer absolute address register
; 141
; 142           rbuf.adrs       =04     ;MEIS receiver buffer address register
; 143           rbuf.data       =05     ;MEIS receiver buffer data register
; 144           rbuf.status     =06     ;MEIS receiver buffer status register
; 145           version         =07     ;MEIS microcode version register
; 146
; 147           header          =10     ;MEIS packet header count register
; 148           trailer         =11     ;MEIS packet trailer register, transmit
; 149           slave.id        =12     ;MEIS slave id register
; 150           old.status.reg  =13     ;previous status register (for ATA detect)
; 151
; 152           bit00           =14     ;bit 00 = 1
; 153           bit03           =15     ;bit 03 = 1
; 154           bit12           =16     ;bit 12 = 1
; 155           temp            =17     ;temporary holding register
; 156
; 157
```

```
; 158   ; 2901 function field
; 159
; 160   ALUFN/=47,6,55,D
; 161           addAQ   =00
; 162           addAB   =01
; 163           addZQ   =02
; 164           addZB   =03
; 165           addZA   =04
; 166           addDA   =05
; 167
; 168           subBA   =11
; 169           subBZ   =13
; 170           subAZ   =14
; 171           subAQ   =20
; 172           subAB   =21
; 173
; 174           bisAQ   =30
; 175           bisAB   =31
; 176           Q       =32
; 177           B       =33
; 178           A       =34
; 179           D       =37
; 180
; 181           andAB   =41
; 182           andDA   =45
; 183           Z       =47
; 184           bicAB   =51
; 185           exnAB   =71
; 186           notA    =74
; 187           notD    =77
; 188
; 189   ;2901 destination field
; 190
; 191   ALUDEST/=1,3,63,D       ;BITS 61,62,63 IN REX.MIC ARE MOVED IN WREX TO 63,48,49
; 192           Qreg    =0
; 193           nop     =1
; 194           Brama   =2
; 195           Bram    =3
; 196           BQdown  =4
; 197           Bdown   =5
; 198           BQup    =6
; 199           Bup     =7
; 200
; 201   ;2901 ALU control
; 202
; 203   CARRY/=0,1,60,D         ;BIT 60 IN REX.MIC IS MOVED IN WREX TO 62
; 204
; 205
```

```
; 206   ; Status control
; 207
; 208   go/=1,1,71,D
; 209           F=1
; 210           T=0
; 211   go                          "go/T"
; 212   comp_error_L/=1,1,70,D
; 213           F=1
; 214           T=0
; 215   comp_error_L                "comp_error_L/T"
; 216   rx_pakt_enb_0/=1,1,69,D
; 217           F=1
; 218           T=0
; 219   rx_pakt_enb_0               "rx_pakt_enb_0/T"
; 220   rx_pakt_enb_1/=1,1,68,D
; 221           F=1
; 222           T=0
; 223   rx_pakt_enb_1               "rx_pakt_enb_1/T"
; 224   mbc_write_mode/=1,1,67,D
; 225           F=1
; 226           T=0
; 227   mbc_write_mode              "mbc_write_mode/T"
; 228
; 229   STATUS/=0,1,32,D
; 230           F=0
; 231           T=1
; 232   clear_status                "STATUS/F"
; 233   set_status                  "STATUS/T"
; 234
```

```
; 235   ; Rcon request control
; 236   ;Note that the Network Interface board microcode uses the inverted octal value
; 237   ;of each Rcon Request code.
; 238
; 239   RCON_SET_REQ/=1,1,22,D
; 240           F=1
; 241           T=0
; 242
; 243   RCON_REQ/=0,3,66,D
; 244
; 245   ;THESE REQUIRE AN XPORT LOAD,
; 246   end_mass_write_trailer=00
; 247   end_mass_write_trailer  "rcon_req/end_mass_write_trailer,RCON_SET_REQ/T"
; 248
; 249   ether_adrs_read=01
; 250   ether_adrs_read         "rcon_req/ether_adrs_read,RCON_SET_REQ/T"
; 251
; 252   flush_tx_bufr=02
; 253   flush_tx_bufr           "rcon_req/flush_tx_bufr,RCON_SET_REQ/T"
; 254
; 255   flush_rx_pakt=03
; 256   flush_rx_pakt           "rcon_req/flush_rx_pakt,RCON_SET_REQ/T"
; 257
; 258   drive_clear=04
; 259   drive_clear             "rcon_req/drive_clear,RCON_SET_REQ/T"
; 260
; 261   bufr_adrs_load=05
; 262   bufr_adrs_load          "rcon_req/bufr_adrs_load,RCON_SET_REQ/T"
; 263
; 264   bufr_data_load=06
; 265   bufr_data_load          "rcon_req/bufr_data_load,RCON_SET_REQ/T"
; 266
; 267   ether_adrs_load=07
; 268   ether_adrs_load         "rcon_req/ether_adrs_load,RCON_SET_REQ/T"
; 269
; 270   ;THESE REQUIRE AN XPORT READ REQ,
; 271   pakt_status_read=0
; 272   pakt_status_read        "RCON_REQ/pakt_status_read,RCON_SET_REQ/T"
; 273
; 274   slide_read=1
; 275   slide_read              "RCON_REQ/slide_read,RCON_SET_REQ/T"
; 276
; 277   start_mass_read=2
; 278   start_mass_read         "RCON_REQ/start_mass_read,RCON_SET_REQ/T"
; 279
; 280   end_mass_write=3
; 281   end_mass_write          "RCON_REQ/end_mass_write,RCON_SET_REQ/T"
; 282
; 283   bufr_ptr_read=4
; 284   bufr_ptr_read           "RCON_REQ/bufr_ptr_read,RCON_SET_REQ/T"
; 285
; 286   end_mass_read=5
; 287   end_mass_read           "RCON_REQ/end_mass_read,RCON_SET_REQ/T"
; 288
```

```
; 289   ; Rbus read/load control
; 290
; 291   asr_read/=1,1,31,D
; 292           F=1
; 293           T=0
; 294   asr_read                  "asr_read/T"
; 295   biker_read/=0,1,30,D
; 296           F=0
; 297           T=1
; 298   biker_read                "biker_read/T"
; 299   bmode_read/=1,1,29,D
; 300           F=1
; 301           T=0
; 302   bmode_read                "bmode_read/T"
; 303   cmax_read/=1,1,28,D
; 304           F=1
; 305           T=0
; 306   cmax_read                 "cmax_read/T"
; 307   maint_read/=1,1,27,D
; 308           F=1
; 309           T=0
; 310   maint_read                "maint_read/T"
; 311   ralu_read/=1,1,26,D
; 312           F=1
; 313           T=0
; 314   continue                  "ralu_read/F"   ;no operation
; 315   ralu_read                 "ralu_read/T"
; 316   status_read/=1,1,25,D
; 317           F=1
; 318           T=0
; 319   status_read               "status_read/T"
; 320   swreg_read/=1,1,24,D
; 321           F=1
; 322           T=0
; 323   swreg_read                "swreg_read/T"
; 324   dpc_read/=1,1,59,D
; 325           F=1
; 326           T=0
; 327   dpc_read                  "dpc_read/T"
; 328   xport_read_req/=1,1,58,D
; 329           F=1
; 330           T=0
; 331   xport_read_req            "xport_read_req/T"
; 332   xport_read/=1,1,57,D
; 333           F=1
; 334           T=0
; 335   xport_read                "xport_read/T"
; 336   xport_load/=1,1,56,D
; 337           F=1
; 338           T=0
; 339   xport_load                "xport_load/T"
; 340   biker_load/=0,1,39,D
; 341           F=0
; 342           T=1
; 343   biker_load                "biker_load/T"
; 344   bheadr_load/=0,1,38,D
; 345           F=0
; 346           T=1
; 347   bheadr_load               "bheadr_load/T"
; 348   bmode_load/=1,1,37,D
; 349           F=1
```

```
; 350           T=0
; 351   bmode_load                "bmode_load/T"
; 352   cmax_load/=1,1,36,D
; 353           F=1
; 354           T=0
; 355   cmax_load                 "cmax_load/T"
; 356   maint_load/=1,1,35,D
; 357           F=1
; 358           T=0
; 359   maint_load                "maint_load/T"
; 360
```

```
; 361   ; Other function bits
; 362
; 363   set_ebl/=1,1,21,D
; 364           F=1
; 365           T=0
; 366   set_ebl          "set_ebl/T"
; 367   set_tra/=0,1,20,D
; 368           F=0
; 369           T=1
; 370   set_tra          "set_tra/T"
; 371   enb_ata/=0,1,19,D
; 372           F=0
; 373           T=1
; 374   enb_ata          "enb_ata/T"
; 375   ata_clear_enb/=1,1,18,D
; 376           F=1
; 377           T=0
; 378   ata_clear_enb            "ata_clear_enb/T"
; 379   dpa_err_clear/=1,1,17,D
; 380           F=1
; 381           T=0
; 382   dpa_err_clear            "dpa_err_clear/T"
; 383   Id_sel/=0,1,33,D
; 384           DRIVE=0
; 385           SERIAL=1
; 386   serial_sel               "ID_SEL/SERIAL"
; 387   drive_sel                "ID_SEL/DRIVE"
; 388   CPA_CLK/=0,1,23,D
; 389           F=0
; 390           T=1
; 391   CPA_CLK          "CPA_CLK/T"
; 392   rmr_test_go/=1,1,9,D
; 393           F=1
; 394           T=0
; 395   rmr_test_go      "rmr_test_go/T"
; 396   rmr_load/=1,1,16,D
; 397           F=1
; 398           T=0
; 399   rmr_load         "rmr_load/T"
; 400
```

```
                                    ; 401   .TOC "Initialization"
                                    ; 402
                                    ; 403           ;Clear Go, the Receiver Packet Enables for both Network Interface boards,
                                    ; 404           ;MBC Write Mode, and the Control register.
                                    ; 405   Init:   clear_status, go, rx_pakt_enb_0, rx_pakt_enb_1, mbc_write_mode, ralu_read,
U 0000, 0014,0346,6721,6017,1177,1402   ; 406            ALUA/temp,ALUB/control.reg,ALUFN/Z,ALUDEST/Bram
                                    ; 407
                                    ; 408           ;Clear the DPA flip-flop and the Composite Error bit in the Status register
                                    ; 409           ;Clear the Error, Maintenance, Byte Count, Mode, and Header registers
                                    ; 410           ;Clear the control massbus transceiver (CMAX) output port
                                    ; 411           dpa_err_clear, set_status, comp_error_L,
                                    ; 412            ralu_read, maint_load, biker_load, bmode_load, bheadr_load, cmax_load,
U 0001, 0024,0246,6730,1437,1177,1435   ; 413            ALUA/temp,ALUB/error.reg,ALUFN/Z,ALUDEST/Bram
                                    ; 414
                                    ; 415           ;Clear ATA, no matter which device number we are.
                                    ; 416           ralu_read, ata_clear_enb,
U 0002, 0034,0306,6721,6377,1637,0437   ; 417            ALUA/temp,ALUB/temp,ALUFN/exnAB,ALUDEST/nop
                                    ; 418           ;Clear the Receiver Buffer Data register
U 0003, 0044,0346,7721,6137,1177,1437   ; 419            ALUA/temp,ALUB/rbuf.data,ALUFN/Z,ALUDEST/Bram
                                    ; 420           ;Clear the Receiver Buffer Status register
U 0004, 0054,0346,7721,6157,1177,1437   ; 421            ALUA/temp,ALUB/rbuf.status,ALUFN/Z,ALUDEST/Bram
                                    ; 422           ;Clear the Diagnostic (Q-reg) register
U 0005, 0064,0346,7721,6377,1177,0037   ; 423            ALUA/temp,ALUB/temp,ALUFN/Z,ALUDEST/Qreg
                                    ; 424           ;Clear the Trailer register
U 0006, 0074,0346,7721,6237,1177,1437   ; 425            ALUA/temp,ALUB/trailer,ALUFN/Z,ALUDEST/Bram
                                    ; 426           ;Clear the Ethernet Address register
U 0007, 0104,0346,7721,6057,1177,1437   ; 427            ALUA/temp,ALUB/ether.adrs,ALUFN/Z,ALUDEST/Bram
                                    ; 428
                                    ; 429           ;Add 1 to error.reg (=0) and store the result as bit00
U 0010, 0114,0346,6721,6301,0117,5437   ; 430           ralu_read, ALUA/error.reg,ALUB/bit00,ALUFN/addZA,ALUDEST/Bram,CARRY/1
                                    ; 431           ;Create bit03
U 0011, 0124,0346,6721,6334,0717,3437   ; 432           ralu_read, ALUA/bit00,ALUB/bit03,ALUFN/A,ALUDEST/Bup
U 0012, 0134,0346,6721,6337,0677,3437   ; 433           ralu_read, ALUA/temp,ALUB/bit03,ALUFN/B,ALUDEST/Bup
U 0013, 0144,0346,6721,6337,0677,3437   ; 434           ralu_read, ALUA/temp,ALUB/bit03,ALUFN/B,ALUDEST/Bup
                                    ; 435           ;Create bit12
U 0014, 0154,0346,6721,6354,0717,2437   ; 436           ralu_read, ALUA/bit00,ALUB/bit12,ALUFN/A,ALUDEST/Bdown
U 0015, 0164,0346,6721,6357,0677,2437   ; 437           ralu_read, ALUA/temp,ALUB/bit12,ALUFN/B,ALUDEST/Bdown
U 0016, 0174,0346,6721,6357,0677,2437   ; 438           ralu_read, ALUA/temp,ALUB/bit12,ALUFN/B,ALUDEST/Bdown
U 0017, 0204,0346,6721,6357,0677,2437   ; 439           ralu_read, ALUA/temp,ALUB/bit12,ALUFN/B,ALUDEST/Bdown
                                    ; 440
                                    ; 441           ;Set up the field for the RMR address register (see RMR: )
                                    ; 442           ;This combination of ALUB/A yields 374 as the RMR address
U 0020, 0214,0346,7721,6374,1177,0437   ; 443   RMRLD:  ALUB/17,ALUA/14
                                    ; 444           ;Now load the RMR address register with the RMR address (374)
U 0021, 0224,0146,7721,6377,1177,0437   ; 445           rmr_load
                                    ; 446
                                    ; 447           ;Start creating the microcode version register
                                    ; 448           ;First, let version = 001000
U 0022, 0234,0346,6721,6176,0717,2437   ; 449           ralu_read, ALUA/bit12,ALUB/version,ALUFN/A,ALUDEST/Bdown
U 0023, 0244,0346,6721,6177,0677,2437   ; 450           ralu_read, ALUA/temp,ALUB/version,ALUFN/B,ALUDEST/Bdown
U 0024, 0254,0346,6721,6177,0677,2437   ; 451           ralu_read, ALUA/temp,ALUB/version,ALUFN/B,ALUDEST/Bdown
                                    ; 452           ;Now add the version register to itself N times so that the Major Revision
                                    ; 453           ;Version field equals N.
                                    ; 454           ;<if used, the code would be:>
                                    ; 455           ;<ralu_read, ALUA/version,ALUB/version,ALUFN/addAB,ALUDEST/Bram,CARRY/0>
                                    ; 456           ;Now increment the version register M times so that the Minor Revision
                                    ; 457           ;field equals M.
U 0025, 0264,0346,6721,6177,0077,5437   ; 458           ralu_read, ALUA/temp,ALUB/version,ALUFN/addZB,ALUDEST/Bram,CARRY/1
U 0026, 0274,0346,6721,6177,0077,5437   ; 459           ralu_read, ALUA/temp,ALUB/version,ALUFN/addZB,ALUDEST/Bram,CARRY/1
U 0027, 0304,0346,6721,6177,0077,5437   ; 460           ralu_read, ALUA/temp,ALUB/version,ALUFN/addZB,ALUDEST/Bram,CARRY/1
                                    ; 461
```

```
                                        ; 462           ;Request the Slave ID of the selected Network Interface board
U 0030, 0324,0344,7721,6377,1175,0477   ; 463   Dregs:  xport_read_req, slide_read
                                        ; 464   =0
                                        ; 465           ;Loop here until we get it.
                                        ; 466           ;This will write the Xport into the Slave ID register many times.
                                        ; 467           ;Only the last time will matter.
                                        ; 468   Slide:  COND/xport, J/Slide, xport_read,
U 0032, 0324,4346,7721,6257,0773,1437   ; 469            ALUA/temp,ALUB/slave.id,ALUFN/D,ALUDEST/Bram
                                        ; 470           ;Request the Packet Status of the selected Network Interface board
U 0033, 0344,0344,7721,6377,1175,0437   ; 471           xport_read_req, pakt_status_read
                                        ; 472   =
                                        ; 473   =0
                                        ; 474           ;Same sort of loop as above.
                                        ; 475   Pktst:  COND/xport, J/Pktst, xport_read,
U 0034, 0344,4346,7721,6157,0773,1437   ; 476              ALUA/temp,ALUB/rbuf.status,ALUFN/D,ALUDEST/Bram
                                        ; 477           ;And now let us be off to the Idle loop.
U 0035, 0364,4746,7721,6377,1177,0437   ; 478           COND/DEMAND, J/IDLE
                                        ; 479   =
                                        ; 480
```

```
                                    ; 481   .TOC "Idle Loop "
                                    ; 482
                                    ; 483           ;There are two ways to return to Idle:
                                    ; 484           ;1. Via a J/IDLE: this postpones checking DEMAND for a while, in case
                                    ; 485           ;       TRA has not yet cleared it .
                                    ; 486           ;2. Via a COND/DEMAND, J/IDLE: this looks for DEMAND immediately, for
                                    ; 487           ;       returns TWO (2) or more instructions after setting TRA.
                                    ; 488   =0
                                    ; 489           ;No DEMAND (or maybe just no test for it): Test for ACQ errors.
                                    ; 490           ;Read the status register to check for an ATA condition.
                                    ; 491   IDLE:   COND/ACQ1.ACQ0, J/ACQ.X, status_read,
U 0036, 0404,6746,5721,6377,1777,1437   ; 492           ALUA/temp,ALUB/temp,ALUFN/notD,ALUDEST/Bram
                                    ; 493           ;DEMAND: service the appropriate register.
U 0037, 7005,0746,7721,6377,1177,0437   ; 494           COND/REG_SEL, J/R.FIELD
                                    ; 495   =
                                    ; 496   =00
                                    ; 497           ;No ACQ errors: test DEMAND.
                                    ; 498           ;Finish checking the Status register for an ATA condition.
                                    ; 499   ACQ.X:  COND/demand, J/DEM1.X, ralu_read, enb_ata,
U 0040, 0464,4766,6721,6277,0637,0437   ; 500           ALUA/temp,ALUB/old.status.reg,ALUFN/bisAB,ALUDEST/nop
                                    ; 501           ;ACQ0 error: go to ACQ0.1 to test DEMAND and check STATUS.
                                    ; 502           ;Set Acqusition Error 0 (bit 12) the Error register.
                                    ; 503           ;Set Composite Error in the Status register
                                    ; 504   ACQ0:   J/ACQ0.1, clear_status, comp_error_L, ralu_read,
U 0041, 0444,0346,6721,6036,0637,1435   ; 505           ALUA/bit12,ALUB/error.reg,ALUFN/bisAB,ALUDEST/Bram
                                    ; 506           ;ACQ1 error: go to ACQ1.1 to set the Error register bit and comp error.
                                    ; 507           ;Shift the Error register down one bit (end-around-shift).
                                    ; 508   ACQ1:   J/ACQ1.1, ralu_read,
U 0042, 0314,0346,6721,6021,0677,2437   ; 509           ALUA/error.reg,ALUB/error.reg,ALUFN/B,ALUDEST/Bdown
                                    ; 510           ;ACQ0 and 1 error: Set Acqusition Error 0 (bit 12) the Error register.
                                    ; 511           ;Set Composite Error in the Status register
                                    ; 512           ;Then go to ACQ1 to set Acqusition Error 1.
                                    ; 513   ACQ01:  J/ACQ1, ralu_read, clear_status, comp_error_L,
U 0043, 0424,0346,6721,6036,0637,1435   ; 514           ALUA/bit12,ALUB/error.reg,ALUFN/bisAB,ALUDEST/Bram
                                    ; 515   =
                                    ; 516           ;Set Acqusition Error 1 (bit 13) the Error register by setting bit 12
                                    ; 517           ;in the downshifted Error register and then shifting the register back up.
                                    ; 518           ;Set Composite Error in the Status register
                                    ; 519   ACQ1.1: ralu_read, clear_status, comp_error_L,
U 0031, 0444,0346,6721,6036,0637,3435   ; 520           ALUA/bit12,ALUB/error.reg,ALUFN/bisAB,ALUDEST/Bup
                                    ; 521           ;Test DEMAND
                                    ; 522           ;Finish checking the Status register for an ATA condition.
                                    ; 523   ACQ0.1: COND/demand, J/DEM1.X, ralu_read, enb_ata,
U 0044, 0464,4766,6721,6277,0637,0437   ; 524           ALUA/temp,ALUB/old.status.reg,ALUFN/bisAB,ALUDEST/nop
                                    ; 525   =0
                                    ; 526           ;No DEMAND: branch on GO and MBC WRITE
                                    ; 527           ;The current Status register now becomes the old Status register
                                    ; 528   DEM1.X: COND/GO.MBC_WRITE, J/GO.W.X,
U 0046, 0504,6346,7721,6277,1717,1437   ; 529           ALUA/temp,ALUB/old.status.reg,ALUFN/notA,ALUDEST/Bram
                                    ; 530           ;DEMAND is set: service the appropriate register.
                                    ; 531           ;The current Status register now becomes the old Status register
                                    ; 532           COND/REG_SEL, J/R.FIELD,
U 0047, 7005,0746,7721,6277,1717,1437   ; 533           ALUA/temp,ALUB/old.status.reg,ALUFN/notA,ALUDEST/Bram
                                    ; 534   =
                                    ; 535
```

```
                                    ; 536
                                    ; 537   ;If GO then do the appropriate read or write routine, else return to IDLE.
                                    ; 538
                                    ; 539   =00
U 0050, 0364,4746,7721,6377,1177,0437   ; 540   GO.W.X: COND/DEMAND,J/IDLE
U 0051, 0364,4746,7721,6377,1177,0437   ; 541           COND/DEMAND,J/IDLE
U 0052, 0544,4746,7721,6377,0057,4037   ; 542           COND/DEMAND,J/R.RD,ALUFN/ADDZQ,ALUDEST/QREG,CARRY/1
U 0053, 0644,4746,7721,6377,0057,4037   ; 543           COND/DEMAND,J/W.WRT,ALUFN/ADDZQ,ALUDEST/QREG,CARRY/1
                                    ; 544   =
                                    ; 545   ;Function GO section (GO=1)
                                    ; 546   ;MBC data read section
                                    ; 547
                                    ; 548   =0
                                    ; 549   ;No DEMAND:  See if we need to timeout the transfer (QREG is negative).
                                    ; 550   ; Create the Massbus Abort bit in TEMP just in case
U 0054, 0564,3346,7721,6376,0717,2437   ; 551   R.RD:   COND/NEG,J/R.RDA,ALUA/BIT12,ALUB/TEMP,ALUFN/A,ALUDEST/BDOWN
                                    ; 552
                                    ; 553   ;DEMAND: service the appropriate register.
U 0055, 7005,0746,7721,6377,1177,0437   ; 554           COND/REG_SEL, J/R.FIELD
                                    ; 555   =
                                    ; 556
                                    ; 557   =0
                                    ; 558   ;Transfer has not exceeded timeout period.  Branch on EXC and EOP.
U 0056, 0604,7346,7721,6377,1177,0437   ; 559   R.RDA:  COND/EXC.EOP, J/R.ENDX
                                    ; 560
                                    ; 561   ;Transfer has timed out.  Set bit 11 in the Error Register to signal an abort.
U 0057, 0454,0346,7721,6037,0637,1437   ; 562           J/R.RDB,ALUA/TEMP,ALUB/ERROR.REG,ALUFN/BISAB,ALUDEST/BRAM
                                    ; 563   =
                                    ; 564
                                    ; 565   ;Finish timing out the read.  Set EBL and end transfer.
                                    ; 566   R.RDB:  COND/END_EBL.DEM,J/EBL.LP,SET_EBL,XPORT_READ_REQ,END_MASS_READ,
U 0045, 1044,7740,7721,6217,1175,1677   ; 567            ALUA/TEMP,ALUB/HEADER,ALUFN/Z,ALUDEST/BRAM
                                    ; 568
                                    ; 569   =00
                                    ; 570           ;Nothing doing: return to Idle.
U 0060, 0364,4746,7721,6377,1177,0437   ; 571   R.ENDX: COND/DEMAND, J/IDLE
                                    ; 572           ;(EXC or EBL): set EBL and end the transfer.
                                    ; 573           COND/END_EBL.DEM, J/EBL.LP, set_ebl, xport_read_req, end_mass_read,
U 0061, 1044,7740,7721,6217,1175,1677   ; 574            ALUA/temp,ALUB/header,ALUFN/Z,ALUDEST/Bram
                                    ; 575           COND/END_EBL.DEM, J/EBL.LP, set_ebl, xport_read_req, end_mass_read,
U 0062, 1044,7740,7721,6217,1175,1677   ; 576            ALUA/temp,ALUB/header,ALUFN/Z,ALUDEST/Bram
                                    ; 577           COND/END_EBL.DEM, J/EBL.LP, set_ebl, xport_read_req, end_mass_read,
U 0063, 1044,7740,7721,6217,1175,1677   ; 578            ALUA/temp,ALUB/header,ALUFN/Z,ALUDEST/Bram
                                    ; 579   =
                                    ; 580
```

```
                                    ; 581
                                    ; 582   ;MBC data write section
                                    ; 583
                                    ; 584   =0
                                    ; 585   ;No DEMAND:  See if we need to timeout the transfer (QREG is negative).
                                    ; 586   ; Create the Massbus Abort bit in TEMP just in case
U 0064, 0664,3346,7721,6376,0717,2437   ; 587   W.WRT:  COND/NEG,J/W.WRTA,ALUA/BIT12,ALUB/TEMP,ALUFN/A,ALUDEST/BDOWN
                                    ; 588
                                    ; 589   ;DEMAND: service the appropriate register.
U 0065, 7005,0746,7721,6377,1177,0437   ; 590           COND/REG_SEL,J/R.FIELD
                                    ; 591   =
                                    ; 592
                                    ; 593   =0
                                    ; 594   ;Transfer has not exceeded timeout period.  Branch on EXC or EOP.
U 0066, 0744,7346,7721,6377,1177,0437   ; 595   W.WRTA: COND/EXC.EOP,J/W.ENDX
                                    ; 596
                                    ; 597   ;Transfer has timed out.  Set bit 11 in the Error Register to signal an abort.
U 0067, 0704,0346,7721,6037,0637,1437   ; 598           J/W.WRTB,ALUA/TEMP,ALUB/ERROR.REG,ALUFN/BISAB,ALUDEST/BRAM
                                    ; 599
                                    ; 600   =
                                    ; 601   ;Here to finish timing out the write.  Set EBL and clear header register
U 0070, 1004,2742,7721,6217,1177,1437   ; 602   W.WRTB: COND/DPA,J/W.END,SET_EBL,ALUA/TEMP,ALUB/HEADER,ALUFN/Z,ALUDEST/BRAM
                                    ; 603
                                    ; 604   =00
                                    ; 605           ;No EXC or EOP: check for a DPA error during the Massbus transfer.
                                    ; 606           ;Prepare the appropriate bit for setting DPA in the Error register
                                    ; 607   W.ENDX: COND/DPA, J/W.DPA,
U 0074, 0724,2746,7721,6375,0717,3437   ; 608            ALUA/bit03,ALUB/temp,ALUFN/A,ALUDEST/Bup
                                    ; 609           ;EXC or EOP: set EBL and check DPA for the last time.
                                    ; 610           ;Clear the Header register
                                    ; 611           COND/DPA, J/W.END, set_ebl,
U 0075, 1004,2742,7721,6217,1177,1437   ; 612            ALUA/temp,ALUB/header,ALUFN/Z,ALUDEST/Bram
                                    ; 613           COND/DPA, J/W.END, set_ebl,
U 0076, 1004,2742,7721,6217,1177,1437   ; 614            ALUA/temp,ALUB/header,ALUFN/Z,ALUDEST/Bram
                                    ; 615           COND/DPA, J/W.END, set_ebl,
U 0077, 1004,2742,7721,6217,1177,1437   ; 616            ALUA/temp,ALUB/header,ALUFN/Z,ALUDEST/Bram
                                    ; 617   =
                                    ; 618           ;You end up here (at the W.DPA branch field) if there is no EXC or EOP
                                    ; 619   =0
                                    ; 620           ;No DPA: go to Idle.
U 0072, 0364,4746,7721,6377,1177,0437   ; 621   W.DPA:  COND/DEMAND, J/IDLE
                                    ; 622           ;DPA is set: set the Comp Error bit in the Status Register.
                                    ; 623           ;Set the Data Bus Parity Error bit (bit 04) in the Error register.
                                    ; 624           COND/DEMAND, J/IDLE, clear_status, comp_error_L,
U 0073, 0364,4746,7721,6037,0637,1435   ; 625            ALUA/temp,ALUB/error.reg,ALUFN/bisAB,ALUDEST/Bram
                                    ; 626   =
                                    ; 627           ;You end up here (at the W.END branch field) if there is an EXC or EOP
                                    ; 628   =0
                                    ; 629           ;No DPA: see if a trailer is required.
U 0100, 1024,5346,7721,6377,1177,0437   ; 630   W.END:  COND/add_trailer_L, J/W.TRL
                                    ; 631           ;DPA: see if a trailer is required.
                                    ; 632           ;Set the DPA bit in the Error register
                                    ; 633           COND/add_trailer_L, J/W.TRL, clear_status, comp_error_L,
U 0101, 1024,5346,7721,6037,0637,1435   ; 634            ALUA/temp,ALUB/error.reg,ALUFN/bisAB,ALUDEST/Bram
                                    ; 635   =
                                    ; 636   =0
                                    ; 637           ;Trailer: end the transfer with a trailer and wait for the end of EBL.
                                    ; 638   W.TRL:  COND/END_EBL.DEM, J/EBL.LP, ralu_read, xport_load, end_mass_write_trailer,
U 0102, 1044,7744,6721,6371,0707,0437   ; 639            ALUA/trailer,ALUB/temp,ALUFN/A,ALUDEST/nop
                                    ; 640           ;No trailer: end the transfer and wait for the end of EBL.
U 0103, 1044,7744,7721,6377,1175,0577   ; 641           COND/END_EBL.DEM, J/EBL.LP, xport_read_req, end_mass_write
```

```
                                        ; 642   =
                                        ; 643   =00
                                        ; 644           ;No END EBL or DEMAND: just loop.
U 0104, 1044,7746,7721,6377,1177,0437   ; 645   EBL.LP: COND/END_EBL.DEM, J/EBL.LP
                                        ; 646           ;DEMAND: service the appropriate register.
U 0105, 7005,0746,7721,6377,1177,0437   ; 647           COND/REG_SEL,J/R.FIELD
                                        ; 648           ;END EBL: clear GO and DPA Error, and return to IDLE.
                                        ; 649           COND/DEMAND, J/IDLE, dpa_err_clear, clear_status, go,
U 0106, 0364,4646,7721,6014,1237,1436   ; 650            ALUA/bit00,ALUB/control.reg,ALUFN/bicAB,ALUDEST/Bram
                                        ; 651           ;END EBL and DEMAND: clear go and service the appropriate register.
                                        ; 652           COND/REG_SEL, J/R.FIELD, dpa_err_clear, clear_status, go,
U 0107, 7005,0646,7721,6014,1237,1436   ; 653            ALUA/bit00,ALUB/control.reg,ALUFN/bicAB,ALUDEST/Bram
                                        ; 654   =
                                        ; 655
```

```
                                        ; 656   .TOC "Register read dispatch table"
                                        ; 657
                                        ; 658   =111000000
                                        ; 659   R.FIELD:
                                        ; 660           ;CONTROL REGISTER
                                        ; 661   R..00:  J/IDLE, ralu_read, cmax_load, set_tra,
U 0700, 0364,0356,6721,2360,0717,0437   ; 662            ALUA/control.reg,ALUB/temp,ALUFN/A,ALUDEST/nop
                                        ; 663           ;STATUS REGISTER
U 0701, 1104,0354,5721,2377,1175,0437   ; 664   R..01:  J/R01, status_read, cmax_load, set_tra, xport_read_req, pakt_status_read
                                        ; 665           ;ERROR REGISTER
                                        ; 666   R..02:  J/IDLE, ralu_read, cmax_load, set_tra,
U 0702, 0364,0356,6721,2361,0717,0437   ; 667            ALUA/error.reg,ALUB/temp,ALUFN/A,ALUDEST/nop
                                        ; 668           ;MAINTENANCE REGISTER
U 0703, 0364,0356,7321,2377,1177,0437   ; 669   R..03:  J/IDLE, maint_read, cmax_load, set_tra
                                        ; 670           ;ATTENTION SUMMARY REGISTER : the tra will be blocked by hardware,
                                        ; 671           ;but the set_tra is still required to set the "Serviced" FF.
U 0704, 0364,0356,7701,2377,1177,0437   ; 672   R..04:  J/IDLE, asr_read, cmax_load, set_tra
                                        ; 673           ;BYTE COUNT REGISTER
U 0705, 0364,0356,7761,2377,1177,0437   ; 674   R..05:  J/IDLE, biker_read, cmax_load, set_tra
                                        ; 675           ;DRIVE TYPE REGISTER
U 0706, 0364,0356,3721,2377,1177,0437   ; 676   R..06:  J/IDLE, swreg_read, drive_sel, cmax_load, set_tra
                                        ; 677           ;DATA CONTROLLER PC REGISTER
U 0707, 0364,0356,7721,2377,1176,0437   ; 678   R..07:  J/IDLE, dpc_read, cmax_load, set_tra
                                        ; 679           ;RECEIVER BUFFER ADDRESS REGISTER
                                        ; 680   R..10:  J/IDLE, ralu_read, cmax_load, set_tra,
U 0710, 0364,0356,6721,2364,0717,0437   ; 681            ALUA/rbuf.adrs,ALUB/temp,ALUFN/A,ALUDEST/nop
                                        ; 682           ;RECEIVER BUFFER DATA REGISTER
                                        ; 683           ;Read the rbuf data and then start working towards a buffer adrs load
                                        ; 684           ;by loading the Buffer Address Load function in the Rcon Request port.
                                        ; 685   R..11:  J/R11, ralu_read, cmax_load, set_tra, bufr_adrs_load,
U 0711, 0714,0354,6721,2105,0077,5277   ; 686            ALUA/rbuf.data,ALUB/rbuf.adrs,ALUFN/addZB,ALUDEST/Brama,CARRY/1
                                        ; 687           ;RECEIVER BUFFER STATUS REGISTER
                                        ; 688   R..12:  J/IDLE, ralu_read, cmax_load, set_tra,
U 0712, 0364,0356,6721,2366,0717,0437   ; 689            ALUA/rbuf.status,ALUB/temp,ALUFN/A,ALUDEST/nop
                                        ; 690           ;DATA CONVERSION MODE REGISTER
U 0713, 0364,0356,7621,2377,1177,0437   ; 691   R..13:  J/IDLE, bmode_read, cmax_load, set_tra
                                        ; 692           ;SERIAL NUMBER REGISTER
U 0714, 0364,0356,3725,2377,1177,0437   ; 693   R..14:  J/IDLE, swreg_read, serial_sel, cmax_load, set_tra
                                        ; 694           ;TRAILER REGISTER
                                        ; 695   R..15:  J/IDLE, ralu_read, cmax_load, set_tra,
U 0715, 0364,0356,6721,2371,0717,0437   ; 696            ALUA/trailer,ALUB/temp,ALUFN/A,ALUDEST/nop
                                        ; 697           ;DIAGNOSTIC REGISTER
                                        ; 698   R..16:  J/IDLE, ralu_read, cmax_load, set_tra,
U 0716, 0364,0356,6721,2377,0657,0437   ; 699            ALUA/temp,ALUB/temp,ALUFN/Q,ALUDEST/nop
                                        ; 700           ;RECEIVER BUFFER ABSOLUTE ADDRESS REGISTER
                                        ; 701   R..17:  J/IDLE, ralu_read, cmax_load, set_tra,
U 0717, 0364,0356,6721,2363,0717,0437   ; 702            ALUA/abs.adrs,ALUB/temp,ALUFN/A,ALUDEST/nop
                                        ; 703           ;ETHERNET ADDRESS REGISTER
                                        ; 704   R..20:  J/IDLE, ralu_read, cmax_load, set_tra,
U 0720, 0364,0356,6721,2362,0717,0437   ; 705            ALUA/ether.adrs,ALUB/temp,ALUFN/A,ALUDEST/nop
                                        ; 706           ;HEADER COUNT REGISTER
                                        ; 707   R..21:  J/IDLE, ralu_read, cmax_load, set_tra,
U 0721, 0364,0356,6721,2370,0717,0437   ; 708            ALUA/header,ALUB/temp,ALUFN/A,ALUDEST/nop
                                        ; 709           ;MICROCODE VERSION REGISTER
                                        ; 710   R..22:  J/IDLE, ralu_read, cmax_load, set_tra,
U 0722, 0364,0356,6721,2367,0717,0437   ; 711            ALUA/version,ALUB/temp,ALUFN/A,ALUDEST/nop
                                        ; 712           ;SLAVE ID REGISTER
                                        ; 713   R..23:  J/IDLE, ralu_read, cmax_load, set_tra,
U 0723, 0364,0356,6721,2372,0717,0437   ; 714            ALUA/slave.id,ALUB/temp,ALUFN/A,ALUDEST/nop
                                        ; 715           ;ILLEGAL REGISTERS
                                        ; 716           ;Illegal registers are read as zero
```

```
                                                ; 717   R..24:  J/R24, ralu_read, cmax_load, set_tra,
U 0724, 1134,0356,6721,2377,1177,0437           ; 718            ALUA/temp,ALUB/temp,ALUFN/Z,ALUDEST/nop
                                                ; 719   R..25:  J/R24, ralu_read, cmax_load, set_tra,
U 0725, 1134,0356,6721,2377,1177,0437           ; 720            ALUA/temp,ALUB/temp,ALUFN/Z,ALUDEST/nop
                                                ; 721   R..26:  J/R24, ralu_read, cmax_load, set_tra,
U 0726, 1134,0356,6721,2377,1177,0437           ; 722            ALUA/temp,ALUB/temp,ALUFN/Z,ALUDEST/nop
                                                ; 723   R..27:  J/R24, ralu_read, cmax_load, set_tra,
U 0727, 1134,0356,6721,2377,1177,0437           ; 724            ALUA/temp,ALUB/temp,ALUFN/Z,ALUDEST/nop
                                                ; 725   R..30:  J/R24, ralu_read, cmax_load, set_tra,
U 0730, 1134,0356,6721,2377,1177,0437           ; 726            ALUA/temp,ALUB/temp,ALUFN/Z,ALUDEST/nop
                                                ; 727   R..31:  J/R24, ralu_read, cmax_load, set_tra,
U 0731, 1134,0356,6721,2377,1177,0437           ; 728            ALUA/temp,ALUB/temp,ALUFN/Z,ALUDEST/nop
                                                ; 729   R..32:  J/R24, ralu_read, cmax_load, set_tra,
U 0732, 1134,0356,6721,2377,1177,0437           ; 730            ALUA/temp,ALUB/temp,ALUFN/Z,ALUDEST/nop
                                                ; 731   R..33:  J/R24, ralu_read, cmax_load, set_tra,
U 0733, 1134,0356,6721,2377,1177,0437           ; 732            ALUA/temp,ALUB/temp,ALUFN/Z,ALUDEST/nop
                                                ; 733   R..34:  J/R24, ralu_read, cmax_load, set_tra,
U 0734, 1134,0356,6721,2377,1177,0437           ; 734            ALUA/temp,ALUB/temp,ALUFN/Z,ALUDEST/nop
                                                ; 735   R..35:  J/R24, ralu_read, cmax_load, set_tra,
U 0735, 1134,0356,6721,2377,1177,0437           ; 736            ALUA/temp,ALUB/temp,ALUFN/Z,ALUDEST/nop
                                                ; 737   R..36:  J/R24, ralu_read, cmax_load, set_tra,
U 0736, 1134,0356,6721,2377,1177,0437           ; 738            ALUA/temp,ALUB/temp,ALUFN/Z,ALUDEST/nop
                                                ; 739   R..37:  J/R24, ralu_read, cmax_load, set_tra,
U 0737, 1134,0356,6721,2377,1177,0437           ; 740            ALUA/temp,ALUB/temp,ALUFN/Z,ALUDEST/nop
                                                ; 741   =
                                                ; 742
```

```
                                    ; 743   .TOC "Register write dispatch table"
                                    ; 744
                                    ; 745   =111100000
                                    ; 746           ;CONTROL REGISTER
U 0740, 1170,0347,7721,6377,1177,0437   ; 747   R..40:  rmr_test_go, J/R40, cpa_clk
                                    ; 748           ;STATUS REGISTER
U 0741, 1320,0347,7721,6377,1177,0437   ; 749   R..41:  rmr_test_go, J/R41, cpa_clk
                                    ; 750           ;ERROR REGISTER
U 0742, 1330,0347,7721,6377,1177,0437   ; 751   R..42:  rmr_test_go, J/R42, cpa_clk
                                    ; 752           ;MAINTENANCE REGISTER
U 0743, 1354,0347,7721,6377,1177,0437   ; 753   R..43:  J/R43, cpa_clk
                                    ; 754           ;ATTENTION SUMMARY REGISTER
U 0744, 1464,0347,7721,6377,1177,0437   ; 755   R..44:  J/R44, cpa_clk
                                    ; 756           ;BYTE COUNT REGISTER
U 0745, 1470,0347,7721,6377,1177,0437   ; 757   R..45:  rmr_test_go, J/R45, cpa_clk
                                    ; 758           ;DRIVE TYPE REGISTER
U 0746, 1320,0347,7721,6377,1177,0437   ; 759   R..46:  rmr_test_go, J/R41, cpa_clk
                                    ; 760           ;DATA CONTROLLER PC REGISTER
U 0747, 1320,0347,7721,6377,1177,0437   ; 761   R..47:  rmr_test_go, J/R41, cpa_clk
                                    ; 762           ;RECEIVER BUFFER ADDRESS REGISTER
U 0750, 1500,0345,7721,6377,1177,0677   ; 763   R..50:  rmr_test_go, J/R50, bufr_adrs_load, cpa_clk
                                    ; 764           ;RECEIVER BUFFER DATA REGISTER
U 0751, 1540,0345,7721,6377,1177,0737   ; 765   R..51:  rmr_test_go, J/R51, bufr_data_load, cpa_clk
                                    ; 766           ;RECEIVER BUFFER STATUS REGISTER
U 0752, 1320,0347,7721,6377,1177,0437   ; 767   R..52:  rmr_test_go, J/R41, cpa_clk
                                    ; 768           ;DATA CONVERSION MODE REGISTER
U 0753, 1640,0347,7721,6377,1177,0437   ; 769   R..53:  rmr_test_go, J/R53, cpa_clk
                                    ; 770           ;SERIAL NUMBER REGISTER
U 0754, 1320,0347,7721,6377,1177,0437   ; 771   R..54:  rmr_test_go, J/R41, cpa_clk
                                    ; 772           ;TRAILER REGISTER
U 0755, 1650,0347,7721,6377,1177,0437   ; 773   R..55:  rmr_test_go, J/R55, cpa_clk
                                    ; 774           ;DIAGNOSTIC REGISTER
U 0756, 1700,0347,7721,6377,1177,0437   ; 775   R..56:  rmr_test_go, J/R56, cpa_clk
                                    ; 776           ;RECEIVER BUFFER ABSOLUTE ADDRESS REGISTER
U 0757, 1710,0347,7721,6377,1177,0437   ; 777   R..57:  rmr_test_go, J/R57, cpa_clk
                                    ; 778           ;ETHERNET ADDRESS REGISTER
U 0760, 1750,0347,7721,6377,1177,0437   ; 779   R..60:  rmr_test_go, J/R60, cpa_clk
                                    ; 780           ;HEADER COUNT REGISTER
U 0761, 2050,0347,7721,6377,1177,0437   ; 781   R..61:  rmr_test_go, J/R61, cpa_clk
                                    ; 782           ;MICROCODE VERSION NUMBER
U 0762, 1320,0347,7721,6377,1177,0437   ; 783   R..62:  rmr_test_go, J/R41, cpa_clk
                                    ; 784           ;SLAVE ID REGISTER
U 0763, 1320,0347,7721,6377,1177,0437   ; 785   R..63:  rmr_test_go, J/R41, cpa_clk
                                    ; 786           ;ILLEGAL REGISTERS
U 0764, 2100,0347,7721,6374,0717,3437   ; 787   R..64:  rmr_test_go, J/R64, cpa_clk, ALUA/bit00,ALUB/temp,ALUFN/A,ALUDEST/Bup
U 0765, 2100,0347,7721,6374,0717,3437   ; 788   R..65:  rmr_test_go, J/R64, cpa_clk, ALUA/bit00,ALUB/temp,ALUFN/A,ALUDEST/Bup
U 0766, 2100,0347,7721,6374,0717,3437   ; 789   R..66:  rmr_test_go, J/R64, cpa_clk, ALUA/bit00,ALUB/temp,ALUFN/A,ALUDEST/Bup
U 0767, 2100,0347,7721,6374,0717,3437   ; 790   R..67:  rmr_test_go, J/R64, cpa_clk, ALUA/bit00,ALUB/temp,ALUFN/A,ALUDEST/Bup
U 0770, 2100,0347,7721,6374,0717,3437   ; 791   R..70:  rmr_test_go, J/R64, cpa_clk, ALUA/bit00,ALUB/temp,ALUFN/A,ALUDEST/Bup
U 0771, 2100,0347,7721,6374,0717,3437   ; 792   R..71:  rmr_test_go, J/R64, cpa_clk, ALUA/bit00,ALUB/temp,ALUFN/A,ALUDEST/Bup
U 0772, 2100,0347,7721,6374,0717,3437   ; 793   R..72:  rmr_test_go, J/R64, cpa_clk, ALUA/bit00,ALUB/temp,ALUFN/A,ALUDEST/Bup
U 0773, 2100,0347,7721,6374,0717,3437   ; 794   R..73:  rmr_test_go, J/R64, cpa_clk, ALUA/bit00,ALUB/temp,ALUFN/A,ALUDEST/Bup
U 0774, 2100,0347,7721,6374,0717,3437   ; 795   R..74:  rmr_test_go, J/R64, cpa_clk, ALUA/bit00,ALUB/temp,ALUFN/A,ALUDEST/Bup
U 0775, 2100,0347,7721,6374,0717,3437   ; 796   R..75:  rmr_test_go, J/R64, cpa_clk, ALUA/bit00,ALUB/temp,ALUFN/A,ALUDEST/Bup
U 0776, 2100,0347,7721,6374,0717,3437   ; 797   R..76:  rmr_test_go, J/R64, cpa_clk, ALUA/bit00,ALUB/temp,ALUFN/A,ALUDEST/Bup
U 0777, 2100,0347,7721,6374,0717,3437   ; 798   R..77:  rmr_test_go, J/R64, cpa_clk, ALUA/bit00,ALUB/temp,ALUFN/A,ALUDEST/Bup
                                    ; 799   =
                                    ; 800
```

```
                                    ; 801   .TOC "Register ucode"
                                    ; 802
                                    ; 803           ;STATUS REGISTER
                                    ; 804   =0
                                    ; 805           ;Wait for the packet status to come back
U 0110, 1104,4346,7721,6377,1177,0437   ; 806   R01:    COND/XPORT, J/R01
                                    ; 807           ;Put the status in the Receiver buffer Status register
                                    ; 808           COND/DEMAND, J/IDLE, xport_read,
U 0111, 0364,4746,7721,6157,0773,1437   ; 809            ALUA/temp,ALUB/rbuf.status,ALUFN/D,ALUDEST/Bram
                                    ; 810   =
                                    ; 811           ;RECEIVER BUFFER DATA REGISTER
                                    ; 812           ;Give a new rbuf adrs to the data board
                                    ; 813   R11:    ralu_read, xport_load,
U 0071, 1124,0346,6721,6364,0707,0437   ; 814            ALUA/rbuf.adrs,ALUB/temp,ALUFN/A,ALUDEST/nop
                                    ; 815           ;After we give the data board the new adrs, we can request a reply
U 0112, 1144,0346,7721,6377,1175,0437   ; 816   R11..1: xport_read_req
                                    ; 817   =0
                                    ; 818           ;Loop until we get an Xport reply
U 0114, 1144,4346,7721,6377,1177,0437   ; 819   R11..2: COND/XPORT, J/R11..2
                                    ; 820           ;Put the Xport data in the Rbuf Data Register and return to Idle
                                    ; 821           COND/DEMAND, J/IDLE, xport_read,
U 0115, 0364,4746,7721,6137,0773,1437   ; 822            ALUA/temp,ALUB/rbuf.data,ALUFN/D,ALUDEST/Bram
                                    ; 823   =
                                    ; 824           ;ILLEGAL REGISTERS
                                    ; 825           ;Create the illegal register bit (bit01) for the Error Register
U 0113, 1164,0346,7721,6374,0717,3437   ; 826   R24:    ALUA/bit00,ALUB/temp,ALUFN/A,ALUDEST/Bup
                                    ; 827           ;Then set the Error Register bit and return to Idle
                                    ; 828           COND/DEMAND, J/IDLE, clear_status, comp_error_L,
U 0116, 0364,4746,7721,6037,0637,1435   ; 829            ALUA/temp,ALUB/error.reg,ALUFN/bisAB,ALUDEST/Bram
                                    ; 830
                                    ; 831           ;CONTROL REGISTER
                                    ; 832           ;Test for a parity error
                                    ; 833           ;Store the massbus data in the Control Register
                                    ; 834   R40:    COND/cpa, J/R40PEX, cmax_read, set_tra,
U 0117, 1204,2356,7521,6017,0777,1437   ; 835            ALUA/temp,ALUB/control.reg,ALUFN/D,ALUDEST/Bram
                                    ; 836   =0
                                    ; 837           ;No parity error: test the error register
                                    ; 838           ;(here we just get set up for checking it)
                                    ; 839           ;(ralu read is diagnostic)
                                    ; 840   R40PEX: J/R40ERT, ralu_read,
U 0120, 1224,0346,6721,6361,0717,0437   ; 841            ALUA/error.reg,ALUB/temp,ALUFN/A,ALUDEST/nop
                                    ; 842           ;Parity error: set the Comp Error bit in the Status Register
                                    ; 843           ;Set the Control Bus Parity Error bit (bit 03) in the Error register
                                    ; 844           ;Go to No Go to exit without performing a function
                                    ; 845           J/R40NGO, clear_status, comp_error_L,
U 0121, 1314,0346,7721,6035,0637,1435   ; 846            ALUA/bit03,ALUB/error.reg,ALUFN/bisAB,ALUDEST/Bram
                                    ; 847   =
                                    ; 848           ;Here we see if the error register is zero
U 0122, 1244,3746,7721,6377,1177,0437   ; 849   R40ERT: COND/zero, J/R40ERX
                                    ; 850   =0
                                    ; 851           ;Error.reg <> zero: go to R40NGO
U 0124, 1314,0346,7721,6377,1177,0437   ; 852   R40ERX: J/R40NGO
                                    ; 853           ;Error.reg = zero: prepare to test GO in the control register
                                    ; 854           J/R40GOT,
U 0125, 1234,0346,7721,6014,1037,0437   ; 855            ALUA/bit00,ALUB/control.reg,ALUFN/andAB,ALUDEST/nop
                                    ; 856   =
                                    ; 857           ;Test to see if the GO bit is set
                                    ; 858           ;Temp= control.reg/2 (Prepare To Vector!)
                                    ; 859   R40GOT: COND/zero,
U 0123, 1264,3746,7721,6360,0717,2437   ; 860            ALUA/control.reg,ALUB/temp,ALUFN/A,ALUDEST/Bdown
                                    ; 861   =0
```

```
                                        ; 862           ;Go <> zero: jump to R40FUN for a function dispatch
                                        ; 863           ;Put the function vector on the RBUS so that it has a cycle to settle
                                        ; 864   R40GOX: J/R40FUN, ralu_read,
U 0126, 1304,0346,6721,6377,0717,0437   ; 865            ALUA/temp,ALUB/temp,ALUFN/A,ALUDEST/nop
                                        ; 866           ;Go = zero: go to R40NGO
U 0127, 1314,0346,7721,6377,1177,0437   ; 867           J/R40NGO
                                        ; 868   =
                                        ; 869           ;Jump to the control register function code
                                        ; 870   R40FUN: COND/FUNC_SEL, J/CF.FIELD,ralu_read,
U 0130, 6406,1346,6721,6377,0717,0437   ; 871            ALUA/temp,ALUB/temp,ALUFN/A,ALUDEST/nop
                                        ; 872           ;Something went wrong, so just clear GO in the control register
                                        ; 873   R40NGO: COND/DEMAND, J/IDLE,
U 0131, 0364,4746,7721,6014,1237,1437   ; 874            ALUA/bit00,ALUB/control.reg,ALUFN/bicAB,ALUDEST/Bram
                                        ; 875
                                        ; 876           ;STATUS REGISTER
                                        ; 877           ;DRIVE TYPE REGISTER (R06)
                                        ; 878           ;DATA CONTROLLER PC REGISTER (R07)
                                        ; 879           ;Check the write parity
U 0132, 2124,2356,7721,6377,1177,0437   ; 880   R41:    COND/cpa, J/PEXIT, set_tra
                                        ; 881
                                        ; 882           ;ERROR REGISTER
                                        ; 883           ;Put the data in the error register
                                        ; 884   R42:    cmax_read, set_tra,
U 0133, 1344,0356,7521,6037,0777,1437   ; 885            ALUA/temp,ALUB/error.reg,ALUFN/D,ALUDEST/Bram
                                        ; 886           ;See if the error register is now zero
U 0134, 1364,3746,7721,6377,1177,0437   ; 887           COND/zero
                                        ; 888   =0
                                        ; 889           ;Error.reg <> zero: set Composite Error
                                        ; 890           ;Check the write parity
U 0136, 2124,2346,7721,6377,1177,0435   ; 891           COND/cpa, J/PEXIT, clear_status, comp_error_L
                                        ; 892           ;Error.reg = zero: just check the write parity
U 0137, 2124,2346,7721,6377,1177,0437   ; 893           COND/cpa, J/PEXIT
                                        ; 894   =
                                        ; 895           ;MAINTENANCE REGISTER
                                        ; 896           ;Put the data into temp
                                        ; 897   R43:    cmax_read, set_tra,
U 0135, 1404,0356,7521,6377,0777,1437   ; 898            ALUA/temp,ALUB/temp,ALUFN/D,ALUDEST/Bram
                                        ; 899           ;Test the Maintenance Mode bit (bit 00) in the Maintenance register
U 0140, 1414,0346,7721,6374,1037,0437   ; 900           ALUA/bit00,ALUB/temp,ALUFN/andAB,ALUDEST/nop
                                        ; 901           ;See if the Maintenance Mode bit is set
U 0141, 1424,3746,7721,6377,1177,0437   ; 902           COND/zero
                                        ; 903   =0
                                        ; 904           ;Maintenance Mode <> zero: check the write parity
U 0142, 1444,2346,7721,6377,1177,0437   ; 905           COND/CPA, J/R43CPA
                                        ; 906           ;Maintenance Mode = zero: clear the Maintenance register
                                        ; 907           ;(also clear temp, so the load at R43CPA also zeros the Maint register)
                                        ; 908           ;Check the write parity
                                        ; 909           COND/CPA, J/R43CPA, ralu_read, maint_load,
U 0143, 1444,2346,6720,6377,1177,1437   ; 910            ALUA/temp,ALUB/temp,ALUFN/Z,ALUDEST/Bram
                                        ; 911   =
                                        ; 912   =0
                                        ; 913           ;No parity error: just transfer temp into the Maintenance register
                                        ; 914   R43CPA: COND/DEMAND, J/IDLE, ralu_read, maint_load,
U 0144, 0364,4746,6720,6377,0717,0437   ; 915            ALUA/temp,ALUB/temp,ALUFN/A,ALUDEST/nop
                                        ; 916           ;Parity error: set the Comp Error bit in the Status Register
                                        ; 917           ;Set the Control Bus Parity Error bit (bit 03) in the Error register
                                        ; 918           ;(and the register is unaffected unless bit00 was 0)
                                        ; 919           COND/DEMAND, J/IDLE, clear_status, comp_error_L,
U 0145, 0364,4746,7721,6035,0637,1435   ; 920            ALUA/bit03,ALUB/error.reg,ALUFN/bisAB,ALUDEST/Bram
                                        ; 921   =
                                        ; 922           ;ATTENTION SUMMARY REGISTER
                                        ; 923           ;Check the write parity
U 0146, 2124,2316,7521,6377,1177,0437   ; 924   R44:    COND/cpa, J/PEXIT, cmax_read, ata_clear_enb, set_tra
                                        ; 925
```

```
                                        ; 926           ;BYTE COUNT REGISTER
                                        ; 927           ;Check the write parity
                                        ; 928   R45:    COND/cpa, J/PEXIT, cmax_read, biker_load, set_tra,
U 0147, 2124,2356,7521,6777,0777,0437   ; 929            ALUA/temp,ALUB/temp,ALUFN/D,ALUDEST/nop
                                        ; 930
```

```
                                    ; 931   ; Register jump table extension
                                    ; 932
                                    ; 933           ;RECEIVER BUFFER ADDRESS REGISTER
                                    ; 934   R50:    cmax_read, xport_load, set_tra,
U 0150, 1514,0356,7521,6117,0767,1437   ; 935            ALUA/temp,ALUB/rbuf.adrs,ALUFN/D,ALUDEST/Bram
                                    ; 936           ;Check the write parity
U 0151, 1524,2346,7721,6377,1175,0437   ; 937           COND/cpa, xport_read_req
                                    ; 938   =0
                                    ; 939           ;Get the data at the new address
                                    ; 940           ;(just like a Receiver Buffer Data register read)
U 0152, 1144,0346,7721,6377,1177,0437   ; 941           J/R11..2
                                    ; 942           ;Parity error: set the Comp Error bit in the Status Register
                                    ; 943           ;Set the Control Bus Parity Error bit (bit 03) in the Error register
                                    ; 944           ;Get the data at the new address
                                    ; 945           ;(just like a Receiver Buffer Data register read)
                                    ; 946           J/R11..2, clear_status, comp_error_L,
U 0153, 1144,0346,7721,6035,0637,1435   ; 947            ALUA/bit03,ALUB/error.reg,ALUFN/bisAB,ALUDEST/Bram
                                    ; 948   =
                                    ; 949           ;RECEIVER BUFFER DATA REGISTER
                                    ; 950           ;Put the data in temp
                                    ; 951   R51:    cmax_read, set_tra,
U 0154, 1554,0356,7521,6377,0777,1437   ; 952            ALUA/temp,ALUB/temp,ALUFN/D,ALUDEST/Bram
                                    ; 953           ;Check the write parity
U 0155, 1564,2346,7721,6377,1177,0437   ; 954           COND/cpa
                                    ; 955   =0
                                    ; 956           ;No parity error: continue
U 0156, 1604,0346,7721,6377,1177,0437   ; 957           J/R51..3
                                    ; 958           ;Parity error: set the Comp Error bit in the Status Register
                                    ; 959           ;Set the Control Bus Parity Error bit (bit 03) in the Error register
                                    ; 960           clear_status, comp_error_L,
U 0157, 1604,0346,7721,6035,0637,1435   ; 961            ALUA/bit03,ALUB/error.reg,ALUFN/bisAB,ALUDEST/Bram
                                    ; 962   =
                                    ; 963           ;See if we are in Maintenance Mode
                                    ; 964   R51..3: maint_read,
U 0160, 1614,0346,7321,6374,1137,0437   ; 965            ALUA/bit00,ALUB/temp,ALUFN/andDA,ALUDEST/nop
U 0161, 1624,3746,7721,6377,1177,0437   ; 966           COND/zero
                                    ; 967   =0
                                    ; 968           ;Maintenance Mode <> zero: write the data to the buffer
                                    ; 969           ;and get the data at the new address
                                    ; 970           ;(just like a Receiver Buffer Data register read)
                                    ; 971           J/R11..1, ralu_read, xport_load,
U 0162, 1124,0346,6721,6377,0707,0437   ; 972            ALUA/temp,ALUB/temp,ALUFN/A,ALUDEST/nop
                                    ; 973           ;Maintenance Mode = zero: do not write the data to the buffer
U 0163, 0364,4746,7721,6377,1177,0437   ; 974           COND/DEMAND, J/IDLE
                                    ; 975   =
                                    ; 976           ;DATA CONVERSION MODE REGISTER
                                    ; 977           ;Check the write parity
U 0164, 1664,2356,7521,4377,1177,0437   ; 978   R53:    COND/cpa, cmax_read, bmode_load, set_tra
                                    ; 979   =0
                                    ; 980           ;No parity error: get the selected board's Slave ID and Packet Status.
U 0166, 0304,0346,7721,6377,1177,0437   ; 981           J/DREGS
                                    ; 982           ;Parity error: set the Comp Error bit in the Status Register
                                    ; 983           ;Set the Control Bus Parity Error bit (bit 03) in the Error register
                                    ; 984           ;Get the selected board's Slave ID and Packet Status.
                                    ; 985           J/DREGS, clear_status, comp_error_L,
U 0167, 0304,0346,7721,6035,0637,1435   ; 986            ALUA/bit03,ALUB/error.reg,ALUFN/bisAB,ALUDEST/Bram
                                    ; 987   =
                                    ; 988           ;TRAILER REGISTER
                                    ; 989           ;Check the write parity and load the Trailer register
                                    ; 990   R55:    COND/cpa, J/PEXIT, cmax_read, set_tra,
U 0165, 2124,2356,7521,6237,0777,1437   ; 991            ALUA/temp,ALUB/trailer,ALUFN/D,ALUDEST/Bram
```

```
                                    ; 992
                                    ; 993           ;DIAGNOSTIC REGISTER
                                    ; 994           ;Check the write parity and load the Diagnostic register
                                    ; 995   R56:    COND/cpa, J/PEXIT, cmax_read, set_tra,
U 0170, 2124,2356,7521,6377,0777,0037 ; 996          ALUA/temp,ALUB/temp,ALUFN/D,ALUDEST/Qreg
                                    ; 997
                                    ; 998           ;RECEIVER BUFFER ABSOLUTE ADDRESS REGISTER
                                    ; 999           ;Note that writing this register causes it to become valid for
                                    ; 1000          ;the subsequent read
                                    ; 1001          ;Check the write parity
U 0171, 1724,2356,7521,6377,1177,0437 ; 1002  R57:    COND/cpa, cmax_read, set_tra
                                    ; 1003  =0
                                    ; 1004          ;No parity error: continue
U 0172, 1744,0346,7721,6377,1177,0437 ; 1005          J/R57..2
                                    ; 1006          ;Parity error: set the Comp Error bit in the Status Register
                                    ; 1007          ;Set the Control Bus Parity Error bit (bit 03) in the Error register
                                    ; 1008          clear_status, comp_error_L,
U 0173, 1744,0346,7721,6035,0637,1435 ; 1009          ALUA/bit03,ALUB/error.reg,ALUFN/bisAB,ALUDEST/Bram
                                    ; 1010  =
                                    ; 1011          ;Make a request to the Network Interface board for the buffer pointer
U 0174, 1764,0344,7721,6377,1175,0637 ; 1012  R57..2: xport_read_req, bufr_ptr_read
                                    ; 1013  =0
                                    ; 1014          ;Loop until we get an Xport reply
U 0176, 1764,4346,7721,6377,1177,0437 ; 1015  R57..3: COND/XPORT, J/R57..3
                                    ; 1016          ;Put the data in the Receiver Absolute Address register
                                    ; 1017          COND/DEMAND, J/IDLE, xport_read,
U 0177, 0364,4746,7721,6077,0773,1437 ; 1018          ALUA/temp,ALUB/abs.adrs,ALUFN/D,ALUDEST/Bram
                                    ; 1019  =
                                    ; 1020
```

```
                                    ; 1021  ; Register jump table extension
                                    ; 1022
                                    ; 1023          ;ETHERNET ADDRESS REGISTER
                                    ; 1024          ;Put the data into the ethernet address register
                                    ; 1025  R60:    cmax_read, set_tra,
U 0175, 2004,0356,7521,6057,0777,1437   ; 1026           ALUA/temp,ALUB/ether.adrs,ALUFN/D,ALUDEST/Bram
                                    ; 1027          ;See if this is an actual write to the ethernet address ram
                                    ; 1028          ;(or is the register just being loaded to check the ram)
U 0200, 2024,3346,7721,6377,1177,0437   ; 1029          COND/neg, J/R60EW
                                    ; 1030  =0
                                    ; 1031          ;Neg = false: the write enable is down so only do an address read
U 0202, 2014,0344,7721,6377,1177,0477   ; 1032  R60EW:  J/R60..3, ether_adrs_read
                                    ; 1033          ;Neg = true: the write enable is up so write the address and then read it
U 0203, 2014,0344,7721,6377,1177,0777   ; 1034          J/R60..3, ether_adrs_load
                                    ; 1035  =
                                    ; 1036          ;The previous instruction only set set up the function, now load the Xport
                                    ; 1037          ;with the ethernet address register initiate it.
                                    ; 1038  R60..3: ralu_read, xport_load,
U 0201, 2044,0346,6721,6362,0707,0437   ; 1039           ALUA/ether.adrs,ALUB/temp,ALUFN/A,ALUDEST/nop
                                    ; 1040          ;Clear the Xport Full flag to initiate the request
U 0204, 2064,0346,7721,6377,1175,0437   ; 1041          xport_read_req
                                    ; 1042  =0
                                    ; 1043          ;Wait here until the D-board responds
U 0206, 2064,4346,7721,6377,1177,0437   ; 1044  R60..5: COND/XPORT,J/R60..5
                                    ; 1045          ;Finally it responds: put the data in the ethernet address register
                                    ; 1046          ;Check the write parity
                                    ; 1047          COND/cpa, J/PEXIT, xport_read,
U 0207, 2124,2346,7721,6057,0773,1437   ; 1048           ALUA/temp,ALUB/ether.adrs,ALUFN/D,ALUDEST/Bram
                                    ; 1049  =
                                    ; 1050
                                    ; 1051          ;HEADER COUNT REGISTER
                                    ; 1052          ;Put the data in the Header Count register
                                    ; 1053          ;Check the write parity
                                    ; 1054  R61:    COND/cpa, J/PEXIT, cmax_read, bheadr_load, set_tra,
U 0205, 2124,2356,7521,7217,0777,1437   ; 1055           ALUA/temp,ALUB/header,ALUFN/D,ALUDEST/Bram
                                    ; 1056
                                    ; 1057          ;ILLEGAL REGISTERS
                                    ; 1058          ;Set Composite Error in the Status register
                                    ; 1059          ;Set the Illegal Register bit (bit 01) in the Error register
                                    ; 1060  R64:    COND/cpa, J/PEXIT, set_tra, clear_status, comp_error_L,
U 0210, 2124,2356,7721,6037,0637,1435   ; 1061           ALUA/temp,ALUB/error.reg,ALUFN/bisAB,ALUDEST/Bram
                                    ; 1062
                                    ; 1063
```

```
                                    ; 1064  ; RMR CODE - NOTE STRATEGIC LOCATION!  CHECK ADDRESSES AFTER ASSEMBLY!
                                    ; 1065
                                    ; 1066          ;Due to the hardware, the branch to RMR defines only Rseqr Adrs bits
                                    ; 1067          ;02 through 08. Bits 00 and 01 are undefined. Therefore, there are 4
                                    ; 1068          ;locations that are RMR. In addition, the msb of RMR must be the same
                                    ; 1069          ;as all of the J/ fields in the register write dispatch field, so RMR
                                    ; 1070          ;should be placed near the code they point to.
                                    ; 1071
                                    ; 1072  ;011 111 100 = 374 = the alu_ab field for the rmr_load (see RMRLD:)
                                    ; 1073
                                    ; 1074  =011111100
                                    ; 1075
                                    ; 1076          ;Jump to RMR parity error test
                                    ; 1077          ;Create bit 02 in temp
                                    ; 1078  RMR:    J/RMRPET,
U 0374, 2114,0346,7721,6375,0717,2437   ; 1079           ALUA/bit03,ALUB/temp,ALUFN/A,ALUDEST/Bdown
                                    ; 1080          J/RMRPET,
U 0375, 2114,0346,7721,6375,0717,2437   ; 1081           ALUA/bit03,ALUB/temp,ALUFN/A,ALUDEST/Bdown
                                    ; 1082          J/RMRPET,
U 0376, 2114,0346,7721,6375,0717,2437   ; 1083           ALUA/bit03,ALUB/temp,ALUFN/A,ALUDEST/Bdown
                                    ; 1084          J/RMRPET,
U 0377, 2114,0346,7721,6375,0717,2437   ; 1085           ALUA/bit03,ALUB/temp,ALUFN/A,ALUDEST/Bdown
                                    ; 1086  =
                                    ; 1087          ;Test for parity error. Set the Composite Error bit in the Status register
                                    ; 1088          ;Set the Register Modification Refused bit (bit 02) in the Error register
                                    ; 1089  RMRPET: COND/cpa, J/PEXIT, set_tra, clear_status, comp_error_L,
U 0211, 2124,2356,7721,6037,0637,1435   ; 1090           ALUA/temp,ALUB/error.reg,ALUFN/bisAB,ALUDEST/Bram
                                    ; 1091
                                    ; 1092          ;This is the parity error branch field, common to many registers.
                                    ; 1093  =0
                                    ; 1094          ;No parity error: return
U 0212, 0364,0346,7721,6377,1177,0437   ; 1095  PEXIT:  J/IDLE
                                    ; 1096          ;Parity error: set the Comp Error bit in the Status Register
                                    ; 1097          ;Set the Control Bus Parity Error bit (bit 03) in the Error register
                                    ; 1098          J/IDLE, clear_status, comp_error_L,
U 0213, 0364,0346,7721,6035,0637,1435   ; 1099           ALUA/bit03,ALUB/error.reg,ALUFN/bisAB,ALUDEST/Bram
                                    ; 1100  =
                                    ; 1101
```

```
                                    ; 1102
                                    ; 1103  .TOC "CONTROL FUNCTION DISPATCH TABLE"
                                    ; 1104
                                    ; 1105  ;Define a couple of useful macros.  CLEAR_GO drops the GO status line and
                                    ; 1106  ; clears the GO bit in the command register.  ILLEGAL_FUNCTION invokes
                                    ; 1107  ; CLEAR_GO and jumps to the illegal function handler.
                                    ; 1108
                                    ; 1109  CLEAR_GO "CLEAR_STATUS,GO,ALUA/BIT00,ALUB/CONTROL.REG,ALUFN/BICAB,ALUDEST/BRAM"
                                    ; 1110
                                    ; 1111  ILLEGAL_FUNCTION "J/ILF,CLEAR_GO"
                                    ; 1112
                                    ; 1113  =110100000
                                    ; 1114
                                    ; 1115  CF.FIELD:
                                    ; 1116          ;No operation
U 0640, 0364,4746,7721,6014,1237,1436   ; 1117  CF..00: COND/DEMAND,J/IDLE,CLEAR_GO
                                    ; 1118          ;Enable Packet Reception.  Clear Go and branch on selected board.
U 0641, 2144,5746,7721,6014,1237,1436   ; 1119  CF..01: COND/SEL_BOARD_1,J/CF01,CLEAR_GO
                                    ; 1120          ;Disable Packet Reception.  Clear Go and branch on selected board.
U 0642, 2164,5746,7721,6014,1237,1436   ; 1121  CF..02: COND/SEL_BOARD_1,J/CF02,CLEAR_GO
                                    ; 1122          ;Flush Packet
U 0643, 0364,4744,7721,6014,1227,1576   ; 1123  CF..03: COND/DEMAND,J/IDLE,CLEAR_GO,FLUSH_RX_PAKT,XPORT_LOAD
                                    ; 1124          ;MEIS Clear
U 0644, 2204,0344,7721,6014,1227,1636   ; 1125  CF..04: J/CF04,CLEAR_GO,XPORT_LOAD,DRIVE_CLEAR
                                    ; 1126          ;Flush Transmit Buffer
U 0645, 0364,4744,7721,6014,1227,1536   ; 1127  CF..05: COND/DEMAND,J/IDLE,CLEAR_GO,FLUSH_TX_BUFR,XPORT_LOAD
U 0646, 2254,0346,7721,6014,1237,1436   ; 1128  CF..06: ILLEGAL_FUNCTION
U 0647, 2254,0346,7721,6014,1237,1436   ; 1129  CF..07: ILLEGAL_FUNCTION
                                    ; 1130          ;MEIS Preset
U 0650, 2204,0344,7721,6014,1227,1636   ; 1131  CF..10: J/CF04,CLEAR_GO,XPORT_LOAD,DRIVE_CLEAR
U 0651, 2254,0346,7721,6014,1237,1436   ; 1132  CF..11: ILLEGAL_FUNCTION
U 0652, 2254,0346,7721,6014,1237,1436   ; 1133  CF..12: ILLEGAL_FUNCTION
U 0653, 2254,0346,7721,6014,1237,1436   ; 1134  CF..13: ILLEGAL_FUNCTION
U 0654, 2254,0346,7721,6014,1237,1436   ; 1135  CF..14: ILLEGAL_FUNCTION
U 0655, 2254,0346,7721,6014,1237,1436   ; 1136  CF..15: ILLEGAL_FUNCTION
U 0656, 2254,0346,7721,6014,1237,1436   ; 1137  CF..16: ILLEGAL_FUNCTION
U 0657, 2254,0346,7721,6014,1237,1436   ; 1138  CF..17: ILLEGAL_FUNCTION
U 0660, 2254,0346,7721,6014,1237,1436   ; 1139  CF..20: ILLEGAL_FUNCTION
U 0661, 2254,0346,7721,6014,1237,1436   ; 1140  CF..21: ILLEGAL_FUNCTION
U 0662, 2254,0346,7721,6014,1237,1436   ; 1141  CF..22: ILLEGAL_FUNCTION
U 0663, 2254,0346,7721,6014,1237,1436   ; 1142  CF..23: ILLEGAL_FUNCTION
U 0664, 2254,0346,7721,6014,1237,1436   ; 1143  CF..24: ILLEGAL_FUNCTION
U 0665, 2254,0346,7721,6014,1237,1436   ; 1144  CF..25: ILLEGAL_FUNCTION
U 0666, 2254,0346,7721,6014,1237,1436   ; 1145  CF..26: ILLEGAL_FUNCTION
U 0667, 2254,0346,7721,6014,1237,1436   ; 1146  CF..27: ILLEGAL_FUNCTION
                                    ; 1147          ;Transmit Packet.  Put zero in the Qreg to start the transfer timer
U 0670, 2234,0346,7731,6377,1177,0017   ; 1148  CF..30: J/CF30,SET_STATUS,MBC_WRITE_MODE,ALUFN/Z,ALUDEST/QREG
U 0671, 2254,0346,7721,6014,1237,1436   ; 1149  CF..31: ILLEGAL_FUNCTION
U 0672, 2254,0346,7721,6014,1237,1436   ; 1150  CF..32: ILLEGAL_FUNCTION
U 0673, 2254,0346,7721,6014,1237,1436   ; 1151  CF..33: ILLEGAL_FUNCTION
                                    ; 1152          ;Receive Packet.  Put zero in the Qreg to start the transfer timer.
U 0674, 2244,0346,7721,6377,1177,0017   ; 1153  CF..34: J/CF34,CLEAR_STATUS,MBC_WRITE_MODE,ALUFN/Z,ALUDEST/QREG
U 0675, 2254,0346,7721,6014,1237,1436   ; 1154  CF..35: ILLEGAL_FUNCTION
U 0676, 2254,0346,7721,6014,1237,1436   ; 1155  CF..36: ILLEGAL_FUNCTION
U 0677, 2254,0346,7721,6014,1237,1436   ; 1156  CF..37: ILLEGAL_FUNCTION
                                    ; 1157  =
                                    ; 1158
```

```
                                    ; 1159
                                    ; 1160  .TOC "CONTROL FUNCTION MICROCODE"
                                    ; 1161
                                    ; 1162  ;Enable Packet Reception
                                    ; 1163  =0
                                    ; 1164  ;Board 0 selected: enable reception for Network Interface board 0
U 0214, 0364,4746,7731,6377,1177,0433   ; 1165  CF01:   COND/DEMAND, J/IDLE, set_status, rx_pakt_enb_0
                                    ; 1166
                                    ; 1167  ;Board 1 selected: enable reception for Network Interface board 1
U 0215, 0364,4746,7731,6377,1177,0427   ; 1168          COND/DEMAND, J/IDLE, set_status, rx_pakt_enb_1
                                    ; 1169  =
                                    ; 1170
                                    ; 1171  ;Disable Packet Reception
                                    ; 1172  =0
                                    ; 1173  ;Board 0 selected: disable reception for Network Interface board 0
U 0216, 0364,4746,7721,6377,1177,0433   ; 1174  CF02:   COND/DEMAND, J/IDLE, clear_status, rx_pakt_enb_0
                                    ; 1175
                                    ; 1176  ;Board 1 selected: disable reception for Network Interface board 1
U 0217, 0364,4746,7721,6377,1177,0427   ; 1177          COND/DEMAND, J/IDLE, clear_status, rx_pakt_enb_1
                                    ; 1178  =
                                    ; 1179
                                    ; 1180  ;MEIS Clear
                                    ; 1181
                                    ; 1182  ;Clear the error register
U 0220, 2214,0246,7721,6037,1177,1437   ; 1183  CF04:   dpa_err_clear,ALUA/temp,ALUB/error.reg,ALUFN/Z,ALUDEST/Bram
                                    ; 1184
                                    ; 1185  ;Clear the Maintenance register, clear the Diagnostic register
U 0221, 2224,0346,6720,6377,1177,0037   ; 1186          ralu_read, maint_load,ALUA/temp,ALUB/temp,ALUFN/Z,ALUDEST/Qreg
                                    ; 1187
                                    ; 1188  ;Clear the ATA register and Composite Error
                                    ; 1189          COND/DEMAND, J/IDLE, ralu_read, ata_clear_enb, set_status,comp_error_L,
U 0222, 0364,4706,6731,6377,1637,0435   ; 1190           ALUA/temp,ALUB/temp,ALUFN/exnAB,ALUDEST/nop
                                    ; 1191
```

```
                                    ; 1192
                                    ; 1193  ;Transmit Packet
                                    ; 1194
                                    ; 1195  ;Set GO and return to Idle
U 0223, 0364,4646,7731,6377,1177,0436   ; 1196  CF30:   COND/DEMAND,J/IDLE,DPA_ERR_CLEAR,SET_STATUS,GO
                                    ; 1197
                                    ; 1198  ;Receive Packet
                                    ; 1199
                                    ; 1200  ;Set GO and go to Idle
U 0224, 0364,4744,7731,6377,1175,0536   ; 1201  CF34:   COND/DEMAND,J/IDLE,SET_STATUS,GO,XPORT_READ_REQ,START_MASS_READ
                                    ; 1202
                                    ; 1203  ;Illegal Function
                                    ; 1204
                                    ; 1205  ;Set the Comp Error bit in the Status Register, set the Illegal Function bit
                                    ; 1206  ; (bit 00) in the Error register.
                                    ; 1207  ILF:    COND/DEMAND,J/IDLE,CLEAR_STATUS,COMP_ERROR_L,ALUA/BIT00,ALUB/ERROR.REG,
U 0225, 0364,4746,7721,6034,0637,1435   ; 1208           ALUFN/BISAB,ALUDEST/BRAM
                                    ; 1209


; NUMBER OF MICRO WORDS USED:
;       D WORDS= 0
;       U WORDS= 250

        END
```

; CROSS REFERENCE LISTING

```
        BDOWN                   197 #   436     437     438     439     449     450     451     509     551     587     860
                                1079    1081    1083    1085
        BQDOWN                  196 #
        BQUP                    198 #
        BRAM                    195 #   406     413     419     421     425     427     430     458     459     460     469
                                476     492     505     514     529     533     562     567     574     576     578     598
                                602     612     614     616     625     634     650     653     809     822     829     835
                                846     874     885     898     910     920     935     947     952     961     986     991
                                1009    1018    1026    1048    1055    1061    1090    1099    1117    1119    1121    1123
                                1125    1127    1128    1129    1131    1132    1133    1134    1135    1136    1137    1138
                                1139    1140    1141    1142    1143    1144    1145    1146    1149    1150    1151    1154
                                1155    1156    1183    1208
        BRAMA                   194 #   686
        BUP                     199 #   432     433     434     520     608     787     788     789     790     791     792
                                793     794     795     796     797     798     826
        NOP                     193 #   417     500     524     639     662     667     681     689     696     699     702
                                705     708     711     714     718     720     722     724     726     728     730     732
                                734     736     738     740     814     841     855     865     871     900     915     929
                                965     972     1039    1190
        QREG                    192 #   423     542     543     996     1148    1153    1186
(U) ALUFN                       160 #
        A                       178 #   432     436     449     551     587     608     639     662     667     681     689
                                696     702     705     708     711     714     787     788     789     790     791     792
                                793     794     795     796     797     798     814     826     841     860     865     871
                                915     972     1039    1079    1081    1083    1085
        ADDAB                   162 #
        ADDAQ                   161 #
        ADDDA                   166 #
        ADDZA                   165 #   430
        ADDZB                   164 #   458     459     460     686
        ADDZQ                   163 #   542     543
        ANDAB                   181 #   855     900
        ANDDA                   182 #   965
        B                       177 #   433     434     437     438     439     450     451     509
        BICAB                   184 #   650     653     874     1117    1119    1121    1123    1125    1127    1128    1129
                                1131    1132    1133    1134    1135    1136    1137    1138    1139    1140    1141    1142
                                1143    1144    1145    1146    1149    1150    1151    1154    1155    1156
        BISAB                   175 #   500     505     514     520     524     562     598     625     634     829     846
                                920     947     961     986     1009    1061    1090    1099    1208
        BISAQ                   174 #
        D                       179 #   469     476     809     822     835     885     898     929     935     952     991
                                996     1018    1026    1048    1055
        EXNAB                   185 #   417     1190
        NOTA                    186 #   529     533
        NOTD                    187 #   492
        Q                       176 #   699
        SUBAB                   172 #
        SUBAQ                   171 #
        SUBAZ                   170 #
        SUBBA                   168 #
        SUBBZ                   169 #
        Z                       183 #   406     413     419     421     423     425     427     567     574     576     578
                                602     612     614     616     718     720     722     724     726     728     730     732
                                734     736     738     740     910     1148    1153    1183    1186
(U) ASR_READ                    291 #
```

```
        F                       292 #
        T                       293 #   672
(U) ATA_CLEAR_ENB               375 #
        F                       376 #
        T                       377 #   416     924     1189
(U) BHEADR_LOAD                 344 #
        F                       345 #
        T                       346 #   412     1054
(U) BIKER_LOAD                  340 #
        F                       341 #
        T                       342 #   412     928
(U) BIKER_READ                  295 #
        F                       296 #
        T                       297 #   674
(U) BMODE_LOAD                  348 #
        F                       349 #
        T                       350 #   412     978
(U) BMODE_READ                  299 #
        F                       300 #
        T                       301 #   691
(U) CARRY                       203 #   430     458     459     460     542     543     686
(U) CMAX_LOAD                   352 #
        F                       353 #
        T                       354 #   412     661     664     666     669     672     674     676     678     680     685
                                688     691     693     695     698     701     704     707     710     713     717     719
                                721     723     725     727     729     731     733     735     737     739
(U) CMAX_READ                   303 #
        F                       304 #
        T                       305 #   834     884     897     924     928     934     951     978     990     995     1002
                                1025    1054
(U) COMP_ERROR_L                212 #
        F                       213 #
        T                       214 #   411     504     513     519     624     633     828     845     891     919     946
                                960     985     1008    1060    1089    1098    1189    1207
(U) COND                        89 #
        ACQ1.ACQ0               108 #   491
        ADD_TRAILER_L           104 #   630     633
        CPA                     97 #    834     880     891     893     905     909     924     928     937     954     978
                                990     995     1002    1047    1054    1060    1089
        DEMAND                  103 #   478     499     523     540     541     542     543     571     621     624     649
                                808     821     828     873     914     919     974     1017    1117    1123    1127    1165
                                1168    1174    1177    1189    1196    1201    1207
        DPA                     98 #    602     607     611     613     615
        END_EBL.DEM             110 #   566     573     575     577     638     641     645
        EXC.EOP                 109 #   559     595
        FUNC_SEL                92 #    870
        GO.MBC_WRITE            107 #   528
        JUMP_ADRS               90 #
        NEG                     99 #    551     587     1029
        REG_SEL                 91 #    494     532     554     590     647     652
        RMR                     93 #
        SEL_BOARD_1             105 #   1119    1121
        SPARE03                 95 #
        XPORT                   102 #   468     475     806     819     1015    1044
        ZERO                    100 #   849     859     887     902     966
```

```
(U) CPA_CLK                 388 #
        F                   389 #
        T                   390 #   747     749     751     753     755     757     759     761     763     765     767
                            769     771     773     775     777     779     781     783     785     787     788     789
                            790     791     792     793     794     795     796     797     798
(U) DPA_ERR_CLEAR           379 #
        F                   380 #
        T                   381 #   411     649     652     1183    1196
(U) DPC_READ                324 #
        F                   325 #
        T                   326 #   678
(U) ENB_ATA                 371 #
        F                   372 #
        T                   373 #   499     523
(U) GO                      208 #
        F                   209 #
        T                   210 #   405     649     652     1117    1119    1121    1123    1125    1127    1128    1129
                            1131    1132    1133    1134    1135    1136    1137    1138    1139    1140    1141    1142
                            1143    1144    1145    1146    1149    1150    1151    1154    1155    1156    1196    1201
(U) ID_SEL                  383 #
        DRIVE               384 #   676
        SERIAL              385 #   693
(U) J                       85 #
        ACQ.X               491     499 #
        ACQ0                504 #
        ACQ0.1              504     523 #
        ACQ01               513 #
        ACQ1                508 #   513
        ACQ1.1              508     519 #
        CF..00              1117 #
        CF..01              1119 #
        CF..02              1121 #
        CF..03              1123 #
        CF..04              1125 #
        CF..05              1127 #
        CF..06              1128 #
        CF..07              1129 #
        CF..10              1131 #
        CF..11              1132 #
        CF..12              1133 #
        CF..13              1134 #
        CF..14              1135 #
        CF..15              1136 #
        CF..16              1137 #
        CF..17              1138 #
        CF..20              1139 #
        CF..21              1140 #
        CF..22              1141 #
        CF..23              1142 #
        CF..24              1143 #
        CF..25              1144 #
        CF..26              1145 #
        CF..27              1146 #
        CF..30              1148 #
        CF..31              1149 #
```

; CROSS REFERENCE LISTING

```
CF..32          1150 #
CF..33          1151 #
CF..34          1153 #
CF..35          1154 #
CF..36          1155 #
CF..37          1156 #
CF.FIELD        870     1115 #
CF01            1119    1165 #
CF02            1121    1174 #
CF04            1125    1131    1183 #
CF30            1148    1196 #
CF34            1153    1201 #
DEM1.X          499     523     528 #
DREGS           463 #   981     985
EBL.LP          566     573     575     577     638     641     645 #   645
GO.W.X          528     540 #
IDLE            478     491 #   540     541     571     621     624     649     661     666     669     672
                674     676     678     680     688     691     693     695     698     701     704     707
                710     713     808     821     828     873     914     919     974     1017    1095    1098
                1117    1123    1127    1165    1168    1174    1177    1189    1196    1201    1207
ILF             1128    1129    1132    1133    1134    1135    1136    1137    1138    1139    1140    1141
                1142    1143    1144    1145    1146    1149    1150    1151    1154    1155    1156    1207 #
INIT            405 #
PEXIT           880     891     893     924     928     990     995     1047    1054    1060    1089    1095 #
PKTST           475 #   475
R..00           661 #
R..01           664 #
R..02           666 #
R..03           669 #
R..04           672 #
R..05           674 #
R..06           676 #
R..07           678 #
R..10           680 #
R..11           685 #
R..12           688 #
R..13           691 #
R..14           693 #
R..15           695 #
R..16           698 #
R..17           701 #
R..20           704 #
R..21           707 #
R..22           710 #
R..23           713 #
R..24           717 #
R..25           719 #
R..26           721 #
R..27           723 #
R..30           725 #
R..31           727 #
R..32           729 #
R..33           731 #
R..34           733 #
R..35           735 #
```

```
R..36                   737 #
R..37                   739 #
R..40                   747 #
R..41                   749 #
R..42                   751 #
R..43                   753 #
R..44                   755 #
R..45                   757 #
R..46                   759 #
R..47                   761 #
R..50                   763 #
R..51                   765 #
R..52                   767 #
R..53                   769 #
R..54                   771 #
R..55                   773 #
R..56                   775 #
R..57                   777 #
R..60                   779 #
R..61                   781 #
R..62                   783 #
R..63                   785 #
R..64                   787 #
R..65                   788 #
R..66                   789 #
R..67                   790 #
R..70                   791 #
R..71                   792 #
R..72                   793 #
R..73                   794 #
R..74                   795 #
R..75                   796 #
R..76                   797 #
R..77                   798 #
R.ENDX                  559     571 #
R.FIELD                 494     532     554     590     647     652     659 #
R.RD                    542     551 #
R.RDA                   551     559 #
R.RDB                   562     566 #
R01                     664     806 #   806
R11                     685     813 #
R11..1                  816 #   971
R11..2                  819 #   819     941     946
R24                     717     719     721     723     725     727     729     731     733     735     737     739
                        826 #
R40                     747     834 #
R40ERT                  840     849 #
R40ERX                  849     852 #
R40FUN                  864     870 #
R40GOT                  854     859 #
R40GOX                  864 #
R40NGO                  845     852     867     873 #
R40PEX                  834     840 #
R41                     749     759     761     767     771     783     785     880 #
R42                     751     884 #
```

```
        R43                     753     897 #
        R43CPA                  905     909     914 #
        R44                     755     924 #
        R45                     757     928 #
        R50                     763     934 #
        R51                     765     951 #
        R51..3                  957     964 #
        R53                     769     978 #
        R55                     773     990 #
        R56                     775     995 #
        R57                     777     1002 #
        R57..2                  1005    1012 #
        R57..3                  1015 #  1015
        R60                     779     1025 #
        R60..3                  1032    1034    1038 #
        R60..5                  1044 #  1044
        R60EW                   1029    1032 #
        R61                     781     1054 #
        R64                     787     788     789     790     791     792     793     794     795     796     797     798
                                1060 #
        RMR                     1078 #
        RMRLD                   443 #
        RMRPET                  1078    1080    1082    1084    1089 #
        SLIDE                   468 #   468
        W.DPA                   607     621 #
        W.ENDX                  595     607 #
        W.END                   602     611     613     615     630 #
        W.TRL                   630     633     638 #
        W.WRTA                  587     595 #
        W.WRTB                  598     602 #
        W.WRT                   543     587 #
(U) MACRO%
        ASR_READ                294 #   672
        ATA_CLEAR_ENB           378 #   416     924     1189
        BHEADR_LOAD             347 #   412     1054
        BIKER_LOAD              343 #   412     928
        BIKER_READ              298 #   674
        BMODE_LOAD              351 #   412     978
        BMODE_READ              302 #   691
        BUFR_ADRS_LOAD          262 #   685     763
        BUFR_DATA_LOAD          265 #   765
        BUFR_PTR_READ           284 #   1012
        CLEAR_GO                1109 #  1117    1119    1121    1123    1125    1127    1128    1129    1131    1132    1133
                                1134    1135    1136    1137    1138    1139    1140    1141    1142    1143    1144    1145
                                1146    1149    1150    1151    1154    1155    1156
        CLEAR_STATUS            232 #   405     504     513     519     624     633     649     652     828     845     891
                                919     946     960     985     1008    1060    1089    1098    1117    1119    1121    1123
                                1125    1127    1128    1129    1131    1132    1133    1134    1135    1136    1137    1138
                                1139    1140    1141    1142    1143    1144    1145    1146    1149    1150    1151    1153
                                1154    1155    1156    1174    1177    1207
        CMAX_LOAD               355 #   412     661     664     666     669     672     674     676     678     680     685
                                688     691     693     695     698     701     704     707     710     713     717     719
                                721     723     725     727     729     731     733     735     737     739
        CMAX_READ               306 #   834     884     897     924     928     934     951     978     990     995     1002
                                1025    1054
```

; CROSS REFERENCE LISTING

```
        COMP_ERROR_L            215 #   411     504     513     519     624     633     828     845     891     919     946
                                960     985     1008    1060    1089    1098    1189    1207
        CONTINUE                314 #
        CPA_CLK                 391 #   747     749     751     753     755     757     759     761     763     765     767
                                769     771     773     775     777     779     781     783     785     787     788     789
                                790     791     792     793     794     795     796     797     798
        DPA_ERR_CLEAR           382 #   411     649     652     1183    1196
        DPC_READ                327 #   678
        DRIVE_CLEAR             259 #   1125    1131
        DRIVE_SEL               387 #   676
        ENB_ATA                 374 #   499     523
        END_MASS_READ           287 #   566     573     575     577
        END_MASS_WRITE          281 #   641
        END_MASS_WRITE_TRAI     247 #   638
        ETHER_ADRS_LOAD         268 #   1034
        ETHER_ADRS_READ         250 #   1032
        FLUSH_RX_PAKT           256 #   1123
        FLUSH_TX_BUFR           253 #   1127
        GO                      211 #   405     649     652     1117    1119    1121    1123    1125    1127    1128    1129
                                1131    1132    1133    1134    1135    1136    1137    1138    1139    1140    1141    1142
                                1143    1144    1145    1146    1149    1150    1151    1154    1155    1156    1196    1201
        ILLEGAL_FUNCTION        1111 #  1128    1129    1132    1133    1134    1135    1136    1137    1138    1139    1140
                                1141    1142    1143    1144    1145    1146    1149    1150    1151    1154    1155    1156
        MAINT_LOAD              359 #   412     909     914     1186
        MAINT_READ              310 #   669     964
        MBC_WRITE_MODE          227 #   405     1148    1153
        PAKT_STATUS_READ        272 #   471     664
        RALU_READ               315 #   405     412     416     430     432     433     434     436     437     438     439
                                449     450     451     458     459     460     499     504     508     513     519     523
                                638     661     666     680     685     688     695     698     701     704     707     710
                                713     717     719     721     723     725     727     729     731     733     735     737
                                739     813     840     864     870     909     914     971     1038    1186    1189
        RMR_LOAD                399 #   445
        RMR_TEST_GO             395 #   747     749     751     757     759     761     763     765     767     769     771
                                773     775     777     779     781     783     785     787     788     789     790     791
                                792     793     794     795     796     797     798
        RX_PAKT_ENB_0           219 #   405     1165    1174
        RX_PAKT_ENB_1           223 #   405     1168    1177
        SERIAL_SEL              386 #   693
        SET_EBL                 366 #   566     573     575     577     602     611     613     615
        SET_STATUS              233 #   411     1148    1165    1168    1189    1196    1201
        SET_TRA                 370 #   661     664     666     669     672     674     676     678     680     685     688
                                691     693     695     698     701     704     707     710     713     717     719     721
                                723     725     727     729     731     733     735     737     739     834     880     884
                                897     924     928     934     951     978     990     995     1002    1025    1054    1060
                                1089
        SLIDE_READ              275 #   463
        START_MASS_READ         278 #   1201
        STATUS_READ             319 #   491     664
        SWREG_READ              323 #   676     693
        XPORT_LOAD              339 #   638     813     934     971     1038    1123    1125    1127    1131
        XPORT_READ_REQ          331 #   463     471     566     573     575     577     641     664     816     937     1012
                                1041    1201
        XPORT_READ              335 #   468     475     808     821     1017    1047
(U) MAINT_LOAD                  356 #
```

```
        F                       357 #
        T                       358 #   412     909     914     1186
(U) MAINT_READ                  307 #
        F                       308 #
        T                       309 #   669     964
(U) MBC_WRITE_MODE              224 #
        F                       225 #
        T                       226 #   405     1148    1153
(U) RALU_READ                   311 #
        F                       312 #
        T                       313 #   405     412     416     430     432     433     434     436     437     438     439
                                449     450     451     458     459     460     499     504     508     513     519     523
                                638     661     666     680     685     688     695     698     701     704     707     710
                                713     717     719     721     723     725     727     729     731     733     735     737
                                739     813     840     864     870     909     914     971     1038    1186    1189
(U) RCON_REQ                    243 #
        BUFR_ADRS_LOAD          261 #   685     763
        BUFR_DATA_LOAD          264 #   765
        BUFR_PTR_READ           283 #   1012
        DRIVE_CLEAR             258 #   1125    1131
        END_MASS_READ           286 #   566     573     575     577
        END_MASS_WRITE          280 #   641
        END_MASS_WRITE_TRAI     246 #   638
        ETHER_ADRS_LOAD         267 #   1034
        ETHER_ADRS_READ         249 #   1032
        FLUSH_RX_PAKT           255 #   1123
        FLUSH_TX_BUFR           252 #   1127
        PAKT_STATUS_READ        271 #   471     664
        SLIDE_READ              274 #   463
        START_MASS_READ         277 #   1201
(U) RCON_SET_REQ                239 #
        F                       240 #
        T                       241 #   463     471     566     573     575     577     638     641     664     685     763
                                765     1012    1032    1034    1123    1125    1127    1131    1201
(U) RMR_LOAD                    396 #
        F                       397 #
        T                       398 #   445
(U) RMR_TEST_GO                 392 #
        F                       393 #
        T                       394 #   747     749     751     757     759     761     763     765     767     769     771
                                773     775     777     779     781     783     785     787     788     789     790     791
                                792     793     794     795     796     797     798
(U) RX_PAKT_ENB_0               216 #
        F                       217 #
        T                       218 #   405     1165    1174
(U) RX_PAKT_ENB_1               220 #
        F                       221 #
        T                       222 #   405     1168    1177
(U) SET_EBL                     363 #
        F                       364 #
        T                       365 #   566     573     575     577     602     611     613     615
(U) SET_TRA                     367 #
        F                       368 #
        T                       369 #   661     664     666     669     672     674     676     678     680     685     688
                                691     693     695     698     701     704     707     710     713     717     719     721
```

```
                        723     725     727     729     731     733     735     737     739     834     880     884
                        897     924     928     934     951     978     990     995     1002    1025    1054    1060
                        1089
(U) STATUS              229 #
        F               230 #   405     504     513     519     624     633     649     652     828     845     891
                        919     946     960     985     1008    1060    1089    1098    1117    1119    1121    1123
                        1125    1127    1128    1129    1131    1132    1133    1134    1135    1136    1137    1138
                        1139    1140    1141    1142    1143    1144    1145    1146    1149    1150    1151    1153
                        1154    1155    1156    1174    1177    1207
        T               231 #   411     1148    1165    1168    1189    1196    1201
(U) STATUS_READ         316 #
        F               317 #
        T               318 #   491     664
(U) SWREG_READ          320 #
        F               321 #
        T               322 #   676     693
(U) XPORT_LOAD          336 #
        F               337 #
        T               338 #   638     813     934     971     1038    1123    1125    1127    1131
(U) XPORT_READ_REQ      328 #
        F               329 #
        T               330 #   463     471     566     573     575     577     641     664     816     937     1012
                        1041    1201
(U) XPORT_READ          332 #
        F               333 #
        T               334 #   468     475     808     821     1017    1047
```

; LOCATION / LINE NUMBER INDEX
; DCODE LOC'N 0 1 2 3 4 5 6 7

d 0000

; LOCATION / LINE NUMBER INDEX

; UCODE LOC'N	0	1	2	3	4	5	6	7
u 0000	406	413	417	419	421	423	425	427
u 0010	430	432	433	434	436	437	438	439
u 0020	443	445	449	450	451	458	459	460
u 0030	463	520	469	471	476	478	492	494
u 0040	500	505	509	514	524	567	529	533
u 0050	540	541	542	543	551	554	559	562
u 0060	571	574	576	578	587	590	595	598
u 0070	602	814	621	625	608	612	614	616
u 0100	630	634	639	641	645	647	650	653
u 0110	806	809	816	826	819	822	829	835
u 0120	841	846	849	860	852	855	865	867
u 0130	871	874	880	885	887	898	891	893
u 0140	900	902	905	910	915	920	924	929
u 0150	935	937	941	947	952	954	957	961
u 0160	965	966	972	974	978	991	981	986
u 0170	996	1002	1005	1009	1012	1026	1015	1018
u 0200	1029	1039	1032	1034	1041	1055	1044	1048
u 0210	1061	1090	1095	1099	1165	1168	1174	1177
u 0220	1183	1186	1190	1196	1201	1208		
u 0230								
u 0240								
u 0250								
u 0260								
u 0270								
u 0300								
u 0310								
u 0320								
u 0330								
u 0340								
u 0350								
u 0360								
u 0370					1079	1081	1083	1085
u 0400								
u 0410								
u 0420								
u 0430								
u 0440								
u 0450								
u 0460								
u 0470								
u 0500								
u 0510								
u 0520								
u 0530								
u 0540								
u 0550								
u 0560								
u 0570								

; LOCATION / LINE NUMBER INDEX

; UCODE LOC'N	0	1	2	3	4	5	6	7
u 0600								
u 0610								
u 0620								
u 0630								
u 0640	1117	1119	1121	1123	1125	1127	1128	1129
u 0650	1131	1132	1133	1134	1135	1136	1137	1138
u 0660	1139	1140	1141	1142	1143	1144	1145	1146
u 0670	1148	1149	1150	1151	1153	1154	1155	1156
u 0700	662	664	667	669	672	674	676	678
u 0710	681	686	689	691	693	696	699	702
u 0720	705	708	711	714	718	720	722	724
u 0730	726	728	730	732	734	736	738	740
u 0740	747	749	751	753	755	757	759	761
u 0750	763	765	767	769	771	773	775	777
u 0760	779	781	783	785	787	788	789	790
u 0770	791	792	793	794	795	796	797	798

no errors detected
END OF MICRO CODE ASSEMBLY
used 4.50 seconds

; TABLE OF CONTENTS

; cross reference index
; dcode location / line # index
; ucode location / line # index

```
                    ; 1     COMMENT     VALID 00024 PAGES
                    ; 2     C REC  PAGE   DESCRIPTION
                    ; 3     C00001 00001
                    ; 4     C00004 00002    .TITLE "MEIS Barrel Shifter Microcode"
                    ; 5     C00007 00003    .TOC "Field Definitions"
                    ; 6     C00009 00004    Input and Output Port Control
                    ; 7     C00011 00005    .TOC "Bit selection and shifting macros"
                    ; 8     C00018 00006    .TOC "Initialization and Idle loop"
                    ; 9     C00021 00007    .TOC "MBC READ:  HEADER MODE 0 - 16-BIT BINARY TO 16-BIT FORMATTED BINARY"
                    ; 10    C00023 00008    .TOC "MBC READ:  HEADER MODE 1 - 16-BIT BINARY TO 32-BIT FORMATTED BINARY"
                    ; 11    C00025 00009    .TOC "MBC READ:  MODE 0      - 16-BIT BINARY TO 16-BIT FORMATTED BINARY"
                    ; 12    C00027 00010    .TOC "MBC READ:  MODE 1      - 16-BIT BINARY TO 32-BIT FORMATTED BINARY"
                    ; 13    C00031 00011    .TOC "MBC READ:  MODE 2      - 16-BIT BINARY TO 36-BIT BINARY"
                    ; 14    C00037 00012    .TOC "MBC READ:  MODE 3      - BYTE ASCII TO PACKED ASCII"
                    ; 15    C00040 00013    .TOC "MBC READ:  MODE 4      - 16-BIT BINARY TO 16-BIT F.B. WITH BYTE SWAP"
                    ; 16    C00041 00014    .TOC "MBC READ:  MODE 5      - 8-BIT BINARY TO 9-BIT BINARY"
                    ; 17    C00043 00015    .TOC "MBC READ:  END CODE    - ALL READ MODES"
                    ; 18    C00047 00016    .TOC "MBC WRITE: HEADER MODE 0 - 16-BIT FORMATTED BINARY TO 16-BIT BINARY"
                    ; 19    C00049 00017    .TOC "MBC WRITE: HEADER MODE 1 - 32-BIT FORMATTED BINARY TO 16-BIT BINARY"
                    ; 20    C00051 00018    .TOC "MBC WRITE: MODE 0      - 16-BIT FORMATTED BINARY TO 16-BIT BINARY"
                    ; 21    C00053 00019    .TOC "MBC WRITE: MODE 1      - 32-BIT FORMATTED BINARY TO 16-BIT BINARY"
                    ; 22    C00055 00020    .TOC "MBC WRITE: MODE 2      - 36-BIT BINARY TO 16-BIT BINARY"
                    ; 23    C00061 00021    .TOC "MBC WRITE: MODE 3      - PACKED ASCII TO BYTE ASCII"
                    ; 24    C00064 00022    .TOC "MBC WRITE: MODE 4      - 16-BIT F.B. TO 16-BIT BINARY WITH BYTE SWAP"
                    ; 25    C00065 00023    .TOC "MBC WRITE: MODE 5      - 9-BIT FORMATTED BINARY TO 8-BIT BINARY"
                    ; 26    C00066 00024    .TOC "MBC WRITE: END CODE    - ALL WRITE MODES"
                    ; 27    C00077 ENDMK
                    ; 28    C ;
                    ; 29
```

```
; 30    .TITLE "MEIS Barrel Shifter Microcode"
; 31
; 32    ;Write prom files using WBEX1.SAI[MIC,GFS]
; 33
; 34    ;23-NOV-82 Created BEX02 from BEX01 to eliminate deficiencies in 36-bit mode.
; 35
; 36    ;28-DEC-82 Modified BEX02 to handle 32-bit header mode correctly.
; 37
; 38    ;23-MAR-83 Created BEX03 from BEX02: added testing of XIPR1 during MPC Reads
; 39    ;so that the deassertion of DCON XIPR would be detected. The code then goes
; 40    ;to the appropriate end routine and generates transfers of zero-data bytes
; 41    ;until BIKER ZERO is detected.
; 42
; 43    ;28-MAR-83 Created BEX04 from BEX03: Changed code at RX.X and RX.L to load zeros
; 44    ;instead of data from the previous input word. The remainder of a received data
; 45    ;word is now zero instead of semi-random data.
; 46
; 47    ;19-JUL-83 Created BEX05 from BEX04: Changed code in 36-bit binary to 16-bit
; 48    ;binary mode. The XIP_0 branches are now all XIP_1 branches. This makes 1 and
; 49    ;2 (36-bit) word transfers work.
; 50
; 51    ;20-SEP-83 Created BEX06 from BEX05: Changed code in Packed ASCII to Byte Ascii
; 52    ;mode. This makes transmission of all byte counts work.
; 53
; 54    ;21-SEP-83 Created BEX07 from BEX06: Corrected a test in Packed ASCII to Byte
; 55    ;Ascii mode at PABA4.X . A test of XIP_0 is now on the line following an Outreg
; 56    ;Load instead of the same line, to allow for pipeline delays.
; 57
; 58    ;04-OCT-83 Created BEX08 from BEX07: Deleted much redundant code in Read and
; 59    ;Write 16-bit and 16-bit swapped modes (4 places total). Added 9-bit Binary
; 60    ;to/from 8-bit Binary Mode.
; 61
; 62
```

```
; 63    .TOC "Field Definitions"
; 64
; 65    ;Pipeline next-address field
; 66
; 67    J/=0,9,8,+
; 68
; 69    ; Jump Field/Condition selection
; 70
; 71    COND/=17,4,12,D
; 72            MODE            =00
; 73
; 74            WRT.XIP.HDR     =04
; 75
; 76            SRC_ACK.HDR_M   =11
; 77            SRC_ACK.XIP_0   =12
; 78            SRC_ACK.XIP_1   =13
; 79
; 80            HDR_MODE        =14
; 81            LAST_ACK        =15
; 82            OUTREG_MT       =16
; 83            JUMP_ADRS       =17
; 84
; 85            ;S3     S2   S1   S0
; 86            ;--     --   --   --
; 87            ;       X    X    X     SELECTS LSB OF NEXT ADDRESS
; 88            ;X      X               SELECTS NEXT 2 BITS OF
; 89            ;                         NEXT ADDRESS FROM ONE OF
; 90            ;                         FOUR SOURCES:
; 91            ;0      0              MODE  MODE      XXX
; 92            ;0      1              WRT   XIP       XXX
; 93            ;1      0              JUMP  SRC_ACK   XXX
; 94            ;1      1              JUMP  JUMP      XXX
; 95
; 96            ;NOTE THAT S2 DOES DOUBLE DUTY. FOR A GIVEN VALUE
; 97            ;OF S2, THERE ARE 4 POSSIBLE TEST SELECTIONS FOR
; 98            ;BIT 0 OF THE NEXT ADDRESS, AND 2 POSSIBLE TEST
; 99            ;SELECTIONS FOR BITS 1 AND 2 OF THE NEXT ADDRESS.
; 100
; 101
```

```
; 102   ;Input and Output Port Control
; 103
; 104   INREG_LOAD/=1,1,31,D
; 105           F=1
; 106           T=0
; 107   INREG_LOAD      "INREG_LOAD/T"
; 108
; 109   OUTREG_LOAD/=1,1,30,D
; 110           F=1
; 111           T=0
; 112   OUTREG_LOAD     "OUTREG_LOAD/T"
; 113
; 114   END_XFER/=0,1,29,D
; 115           F=0
; 116           T=1
; 117   END_XFER        "END_XFER/T"
; 118
; 119   ;Output port load controls
; 120   LOADL/=1,1,26,D
; 121   LOADL   "LOADL/0"
; 122
; 123   LOADH/=1,1,25,D
; 124   LOADH   "LOADH/0"
; 125
; 126   LOADX/=1,1,24,D
; 127   LOADX   "LOADX/0"
; 128
; 129   ;Bashing control
; 130
; 131   BYTE_SEL/=0,3,15,D
; 132
; 133   BASH/=0,2,28,D
; 134
; 135   ;Bit-Save register control
; 136
; 137   SAVE/=377,8,23,D              ;bit load is active low
; 138   SAVE0   "SAVE/000"            ;bit numbering is Big Endian (msb=bit 0)
; 139   SAVE1   "SAVE/200"            ;SaveN: loading occurs from bit N
; 140   SAVE2   "SAVE/300"            ;       down to bit 7
; 141   SAVE3   "SAVE/340"
; 142   SAVE4   "SAVE/360"
; 143   SAVE5   "SAVE/370"
; 144   SAVE6   "SAVE/374"
; 145   SAVE7   "SAVE/376"
; 146
```

```
; 147   .TOC "Bit selection and shifting macros"
; 148
; 149   ;BXnn.Dm macros are used to select and load bits from a 36-bit word,
; 150   ;consisting of 2 18-bit massbus transfers. "nn" selects the most significant
; 151   ;bit in the word (using PDP-10 nomenclature, bit 00 is msb), while "m" selects
; 152   ;where that bit is loaded in the 8-bit byte builder. The selected bit and all
; 153   ;less significant bits are loaded. The comment at the end of each line gives the
; 154   ;number of sequential bits the permutation provides.
; 155   ;Example: BX00.D1 loads the most significant bit of the input word into the
; 156   ;second most-significant bit of the byte-builder. Lower order bits of the byte-
; 157   ;builder are also loaded from the essentially down-shifted input word. The msb
; 158   ;of the byte-builder remains unchanged, and the least significant bit is garbage
; 159   ;(BX00.D0 only produces 6 usable bits). The result is
; 160
; 161   ;       PP 00 01 02 03 04 05 GG         (where PP=previous value and GG=garbage).
; 162
; 163   BX00.D0         "BYTE_SEL/7,BASH/2,SAVE0"       ; 6 BITS
; 164   BX00.D1         "BYTE_SEL/7,BASH/1,SAVE1"       ; 6 BITS
; 165   BX00.D4         "BYTE_SEL/6,BASH/2,SAVE4"       ; 4 BITS
; 166   BX01.D0         "BYTE_SEL/7,BASH/3,SAVE0"       ; 5 BITS
; 167   BX02.D0         "BYTE_SEL/2,BASH/0,SAVE0"       ; 8 BITS
; 168   BX04.D0         "BYTE_SEL/2,BASH/2,SAVE0"       ; 6 BITS
; 169   BX06.D5         "BYTE_SEL/1,BASH/3,SAVE5"       ; 3 BITS
; 170   BX06.D6         "BYTE_SEL/1,BASH/2,SAVE6"       ; 2 BITS
; 171   BX06.D7         "BYTE_SEL/1,BASH/1,SAVE7"       ; 1 BIT
; 172   BX07.D1         "BYTE_SEL/1,BASH/0,SAVE1"       ; 3 BITS
; 173   BX08.D0         "BYTE_SEL/1,BASH/2,SAVE0"       ; 6 BITS
; 174   BX08.D4         "BYTE_SEL/2,BASH/2,SAVE4"       ; 2 BITS
; 175   BX10.D0         "BYTE_SEL/0,BASH/0,SAVE0"       ; 8 BITS
; 176   BX10.D6         "BYTE_SEL/0,BASH/2,SAVE6"       ; 2 BITS
; 177   BX12.D0         "BYTE_SEL/0,BASH/2,SAVE0"       ; 6 BITS
; 178   BX14.D1         "BYTE_SEL/0,BASH/3,SAVE1"       ; 4 BITS
; 179   BX14.D2         "BYTE_SEL/0,BASH/2,SAVE2"       ; 4 BITS
; 180   BX14.D6         "BYTE_SEL/3,BASH/2,SAVE6"       ; 2 BITS
; 181   BX16.D0         "BYTE_SEL/3,BASH/2,SAVE0"       ; 2 BITS
; 182   BX18.D2         "BYTE_SEL/7,BASH/0,SAVE2"       ; 6 BITS
; 183   BX18.D5         "BYTE_SEL/6,BASH/1,SAVE5"       ; 3 BITS
; 184   BX18.D6         "BYTE_SEL/6,BASH/0,SAVE6"       ; 2 BITS
; 185   BX20.D0         "BYTE_SEL/2,BASH/0,SAVE0"       ; 8 BITS
; 186   BX21.D1         "BYTE_SEL/2,BASH/0,SAVE1"       ; 7 BITS
; 187   BX24.D0         "BYTE_SEL/1,BASH/0,SAVE0"       ; 8 BITS
; 188   BX28.D0         "BYTE_SEL/0,BASH/0,SAVE0"       ; 8 BITS
; 189   BX28.D1         "BYTE_SEL/1,BASH/3,SAVE1"       ; 4 BITS
; 190   BX32.D0         "BYTE_SEL/3,BASH/0,SAVE0"       ; 4 BITS
; 191   BX32.D5         "BYTE_SEL/3,BASH/3,SAVE5"       ; 3 BITS
; 192
; 193
; 194   ;The Bnn.Dm macros are similar, except that the input is considered to be
; 195   ;16-bit and the "nn" numbering is therefore done in PDP-11 fashion (bit 15
; 196   ;is the msb).
; 197
; 198   B00.D0          "BYTE_SEL/3,BASH/3,SAVE0"       ; 1 BITS
; 199   B01.D0          "BYTE_SEL/3,BASH/2,SAVE0"       ; 2 BITS
; 200   B01.D6          "BYTE_SEL/0,BASH/0,SAVE6"       ; 2 BITS
; 201   B02.D6          "BYTE_SEL/3,BASH/3,SAVE6"       ; 2 BITS
; 202   B03.D0          "BYTE_SEL/3,BASH/0,SAVE0"       ; 4 BITS
; 203   B03.D6          "BYTE_SEL/3,BASH/2,SAVE6"       ; 2 BITS
; 204   B03.D7          "BYTE_SEL/3,BASH/1,SAVE7"       ; 1 BIT
; 205   B04.D0          "BYTE_SEL/0,BASH/3,SAVE0"       ; 5 BITS
; 206   B05.D0          "BYTE_SEL/0,BASH/2,SAVE0"       ; 6 BITS
; 207   B05.D6          "BYTE_SEL/1,BASH/0,SAVE6"       ; 2 BITS
; 208   B06.D0          "BYTE_SEL/0,BASH/1,SAVE0"       ; 7 BITS
; 209   B06.D1          "BYTE_SEL/0,BASH/0,SAVE1"       ; 7 BITS
; 210   B06.D4          "BYTE_SEL/1,BASH/1,SAVE4"       ; 3 BITS
; 211   B06.D5          "BYTE_SEL/1,BASH/0,SAVE5"       ; 3 BITS
; 212   B06.D6          "BYTE_SEL/0,BASH/3,SAVE6"       ; 2 BITS
```

```
; 213   B07.D0          "BYTE_SEL/0,BASH/0,SAVE0"       ; 8 BITS
; 214   B07.D6          "BYTE_SEL/0,BASH/2,SAVE6"       ; 2 BITS
; 215   B08.D0          "BYTE_SEL/1,BASH/3,SAVE0"       ; 5 BITS
; 216   B09.D0          "BYTE_SEL/1,BASH/2,SAVE0"       ; 6 BITS
; 217   B09.D6          "BYTE_SEL/2,BASH/0,SAVE6"       ; 2 BITS
; 218   B10.D6          "BYTE_SEL/1,BASH/3,SAVE6"       ; 2 BITS
; 219   B11.D0          "BYTE_SEL/1,BASH/0,SAVE0"       ; 8 BITS
; 220   B11.D6          "BYTE_SEL/1,BASH/2,SAVE6"       ; 2 BITS
; 221   B11.D7          "BYTE_SEL/1,BASH/1,SAVE7"       ; 1 BIT
; 222   B12.D0          "BYTE_SEL/2,BASH/3,SAVE0"       ; 5 BITS
; 223   B13.D0          "BYTE_SEL/2,BASH/2,SAVE0"       ; 6 BITS
; 224   B13.D6          "BYTE_SEL/3,BASH/0,SAVE6"       ; 2 BITS
; 225   B14.D0          "BYTE_SEL/2,BASH/1,SAVE0"       ; 7 BITS
; 226   B14.D1          "BYTE_SEL/2,BASH/0,SAVE1"       ; 6 BITS
; 227   B14.D4          "BYTE_SEL/3,BASH/1,SAVE4"       ; 3 BITS
; 228   B14.D5          "BYTE_SEL/3,BASH/0,SAVE5"       ; 3 BITS
; 229   B14.D6          "BYTE_SEL/2,BASH/3,SAVE6"       ; 2 BITS
; 230   B15.D0          "BYTE_SEL/2,BASH/0,SAVE0"       ; 8 BITS
; 231   B15.D2          "BYTE_SEL/3,BASH/2,SAVE2"       ; 4 BITS
; 232   B15.D4          "BYTE_SEL/3,BASH/0,SAVE4"       ; 4 BITS
; 233   B15.D6          "BYTE_SEL/2,BASH/2,SAVE6"       ; 2 BITS
; 234   B15.D7          "BYTE_SEL/2,BASH/1,SAVE7"       ; 1 BIT
; 235
; 236   ;BYTE_SEL/4 generates a field of zeros, which can then be loaded into
; 237   ;the appropriate section of the output register with a SAVE.
; 238
; 239   BZERO.D0        "BYTE_SEL/4,SAVE0"
; 240   BZERO.D1        "BYTE_SEL/4,SAVE1"
; 241   BZERO.D2        "BYTE_SEL/4,SAVE2"
; 242   BZERO.D4        "BYTE_SEL/4,SAVE4"
; 243   BZERO.D5        "BYTE_SEL/4,SAVE5"
; 244   BZERO.D6        "BYTE_SEL/4,SAVE6"
; 245   BZERO.D7        "BYTE_SEL/4,SAVE7"
; 246
; 247
```

```
                        ; 248   .TOC "Initialization and Idle loop"
                        ; 249
                        ; 250   =000
U 0000, 0002,0377,7060  ; 251   IDLE:   COND/WRT.XIP.HDR,J/IDLE          ;no transfer in progress
U 0001, 0002,0377,7060  ; 252           COND/WRT.XIP.HDR,J/IDLE          ;no transfer in progress
U 0002, 0100,0377,7060  ; 253           COND/MODE,J/MR.X                 ;Read mode: select which one
U 0003, 0304,4377,7060  ; 254           COND/SRC_ACK.HDR_M,J/HR.SEL      ;Read mode with Header: select Header
                        ; 255
U 0004, 0002,0377,7060  ; 256           COND/WRT.XIP.HDR,J/IDLE          ;no transfer in progress
U 0005, 0002,0377,7060  ; 257           COND/WRT.XIP.HDR,J/IDLE          ;no transfer in progress
U 0006, 0200,0377,7060  ; 258           COND/MODE,J/MW.X                 ;Write mode: select which one
U 0007, 0344,4377,7060  ; 259           COND/SRC_ACK.HDR_M,J/HW.SEL      ;Write mode with Header: select Header
                        ; 260   =
                        ; 261   =000
U 0010, 1027,6374,7060  ; 262   MR.X:   J/B.F16,BZERO.D6                  ;select which Read mode
U 0011, 1117,4377,7060  ; 263           J/B.F32
U 0012, 1407,4377,7060  ; 264           J/B.B36
U 0013, 2617,4377,7060  ; 265           J/BA.PA
                        ; 266
U 0014, 3417,6374,7060  ; 267           J/B.S16,BZERO.D6
U 0015, 3547,4377,7060  ; 268           J/B.B09
U 0016, 1407,4377,7060  ; 269           J/B.B36
U 0017, 2617,4377,7060  ; 270           J/BA.PA
                        ; 271   =
                        ; 272   =000
U 0020, 4657,4377,7060  ; 273   MW.X:   J/F16.B                           ;select which Write mode
U 0021, 5007,4377,7060  ; 274           J/F32.B
U 0022, 5257,4377,7060  ; 275           J/B36.B
U 0023, 6427,4377,7060  ; 276           J/PA.BA
                        ; 277
U 0024, 7167,4377,7060  ; 278           J/F16.S
U 0025, 7257,4377,7060  ; 279           J/B09.B
U 0026, 5257,4377,7060  ; 280           J/B36.B
U 0027, 6427,4377,7060  ; 281           J/PA.BA
                        ; 282   =
                        ; 283   =00
U 0030, 0405,4377,7060  ; 284   HR.SEL: COND/SRC_ACK.XIP_1,J/HR16.X              ;HEADER, 16-BIT BINARY READ MODE
U 0031, 0545,4377,7060  ; 285           COND/SRC_ACK.XIP_1,J/HR32.X              ;HEADER, 32-BIT BINARY READ MODE
U 0032, 0405,4377,7060  ; 286           COND/SRC_ACK.XIP_1,J/HR16.X              ;HEADER, 16-BIT BINARY READ MODE
U 0033, 0545,4377,7060  ; 287           COND/SRC_ACK.XIP_1,J/HR32.X              ;HEADER, 32-BIT BINARY READ MODE
                        ; 288   =
                        ; 289   =00
U 0034, 4205,0377,7060  ; 290   HW.SEL: COND/SRC_ACK.XIP_0,J/HW16.X              ;HEADER, 16-BIT BINARY WRITE MODE
U 0035, 4445,0377,7060  ; 291           COND/SRC_ACK.XIP_0,J/HW32.X              ;HEADER, 32-BIT BINARY WRITE MODE
U 0036, 4205,0377,7060  ; 292           COND/SRC_ACK.XIP_0,J/HW16.X              ;HEADER, 16-BIT BINARY WRITE MODE
U 0037, 4445,0377,7060  ; 293           COND/SRC_ACK.XIP_0,J/HW32.X              ;HEADER, 32-BIT BINARY WRITE MODE
                        ; 294   =
                        ; 295
```

```
                        ; 296   .TOC "MBC READ:  HEADER MODE 0 - 16-BIT BINARY TO 16-BIT FORMATTED BINARY"
                        ; 297
                        ; 298   ; THE 2 MSB'S OF EACH 18-BIT OUTPUT BYTE EQUAL ZERO.
                        ; 299
                        ; 300   =00
U 0040, 3657,4377,7060  ; 301   HR16.X: J/RX6X                                    ;NO ACK, NO XIP, ALL DONE
U 0041, 0405,4377,7040  ; 302           COND/SRC_ACK.XIP_1,INREG_LOAD,J/HR16.X   ;NO ACK, XIP
U 0042, 4147,4377,7060  ; 303           J/READ.FAIL                              ;ACK, NO XIP
U 0043, 0447,6374,7040  ; 304           BZERO.D6,INREG_LOAD                      ;ACK, XIP
                        ; 305   =
U 0044, 0457,5000,3060  ; 306           B15.D0,LOADX
U 0045, 0467,4000,5060  ; 307           B07.D0,LOADH
                        ; 308   =0
U 0046, 0467,0377,6060  ; 309   HR16.Z: LOADL,COND/OUTREG_MT,J/HR16.Z
U 0047, 0507,4377,7020  ; 310           OUTREG_LOAD                              ;LOAD WORD AND DECREMENT BIKER
                        ; 311   =
U 0050, 0526,0377,7060  ; 312           COND/HDR_MODE,J/HR16.HDR.X               ;SEE IF WE ARE STILL IN HEADER MODE
                        ; 313   =0
U 0052, 0100,0377,7060  ; 314   HR16.HDR.X:COND/MODE,J/MR.X                       ;END OF HEADER MODE
U 0053, 0405,4377,7060  ; 315           COND/SRC_ACK.XIP_1,J/HR16.X               ;CONTINUE IN HEADER MODE
                        ; 316   =
                        ; 317
```

```
                    ; 318   .TOC "MBC READ:  HEADER MODE 1 - 16-BIT BINARY TO 32-BIT FORMATTED BINARY"
                    ; 319
                    ; 320   ; THE 4 LSB'S OF EACH 36-BIT OUTPUT WORD EQUAL ZERO.
                    ; 321
                    ; 322   =00
U 0054, 3657,4377,7060  ; 323   HR32.X: J/RX6X                                    ;NO ACK, NO XIP
U 0055, 0545,4377,7040  ; 324           COND/SRC_ACK.XIP_1,INREG_LOAD,J/HR32.X  ;NO ACK, XIP
U 0056, 4147,4377,7060  ; 325           J/READ.FAIL                             ;ACK, NO XIP
U 0057, 0517,4377,7040  ; 326           INREG_LOAD                              ;ACK, XIP
                    ; 327   =
U 0051, 0607,5374,7460  ; 328           B15.D6                  ;get 2 bits
U 0060, 0617,5000,3460  ; 329           B13.D0,LOADX            ;get 6 bits
U 0061, 0627,4374,7460  ; 330           B07.D6                  ;get 2 bits
U 0062, 0645,4000,5460  ; 331           B05.D0,LOADH,COND/SRC_ACK.XIP_1,J/HR321.X       ;GET BYTES 2,3
                    ; 332   =00                             ;get 6 bits
U 0064, 4017,4377,7060  ; 333   HR321.X:J/RX6L                                  ;NO ACK, NO XIP
U 0065, 0645,4377,7040  ; 334           COND/SRC_ACK.XIP_1,INREG_LOAD,J/HR321.X ;NO ACK, XIP
U 0066, 4147,4377,7060  ; 335           J/READ.FAIL                             ;ACK, NO XIP
U 0067, 0637,4377,7040  ; 336           INREG_LOAD                              ;ACK, XIP
                    ; 337   =
U 0063, 0707,5374,7460  ; 338           B15.D6                  ;get 2 bits
                    ; 339   =0
U 0070, 0707,0377,6060  ; 340   HR321.Z:LOADL,COND/OUTREG_MT,J/HR321.Z
U 0071, 0727,5774,7020  ; 341           B13.D6,OUTREG_LOAD      ;get 2 bits     ;LOAD WORD 0, BYTE 0
                    ; 342   =
U 0072, 0737,4400,3060  ; 343           B11.D0,LOADX            ;get 8 bits
U 0073, 0747,5400,5060  ; 344           B03.D0,LOADH            ;get 4 bits
U 0074, 0767,6360,7060  ; 345           BZERO.D4                ;get 4 bits of zero
                    ; 346   =0
U 0076, 0767,0377,6060  ; 347   HR322.Z:LOADL,COND/OUTREG_MT,J/HR322.Z
U 0077, 0757,4377,7020  ; 348           OUTREG_LOAD                             ;LOAD WORD 0, BYTE 1
                    ; 349   =
U 0075, 1006,0377,7060  ; 350           COND/HDR_MODE,J/HR32.HDR.X              ;SEE IF WE ARE STILL IN HEADER MODE
                    ; 351   =0
U 0100, 0100,0377,7060  ; 352   HR32.HDR.X:COND/MODE,J/MR.X                     ;END OF HEADER MODE
U 0101, 0545,4377,7060  ; 353           COND/SRC_ACK.XIP_1,J/HR32.X             ;CONTINUE IN HEADER MODE
                    ; 354   =
                    ; 355
```

```
                        ; 356   .TOC "MBC READ:  MODE 0      - 16-BIT BINARY TO 16-BIT FORMATTED BINARY"
                        ; 357
                        ; 358   ; THE 2 MSB'S OF EACH 18-BIT OUTPUT BYTE EQUAL ZERO.
                        ; 359   ; NOTE THAT "INREG_LOAD" IS THE MIO OR MBUS REQUEST FOR MORE DATA !
                        ; 360
U 0102, 1045,4377,7060  ; 361   B.F16:  COND/SRC_ACK.XIP_1,J/BF16.X     ;wait for Bytes 0 and 1
                        ; 362   =00
U 0104, 3657,4377,7060  ; 363   BF16.X: J/RX6X                                  ;NO ACK, NO XIP
U 0105, 1045,4377,7040  ; 364           COND/SRC_ACK.XIP_1,INREG_LOAD,J/BF16.X  ;NO ACK, XIP
U 0106, 4147,4377,7060  ; 365           J/READ.FAIL                             ;ACK, NO XIP
U 0107, 1037,6374,7040  ; 366           BZERO.D6,INREG_LOAD                     ;ACK, XIP
                        ; 367   =                                       ;get 2 bits of zero
U 0103, 1107,5000,3060  ; 368           B15.D0,LOADX                    ;get 8 bits
U 0110, 1127,4000,5060  ; 369           B07.D0,LOADH                    ;get 8 bits
                        ; 370   =0
U 0112, 1127,0377,6060  ; 371   BF16.Z: LOADL,COND/OUTREG_MT,J/BF16.Z   ;wait until the Outreg is empty
U 0113, 1027,4377,7020  ; 372           OUTREG_LOAD,J/B.F16             ;then load it with Word 0, Byte 0
                        ; 373   =
                        ; 374
```

```
                        ; 375   .TOC "MBC READ:  MODE 1      - 16-BIT BINARY TO 32-BIT FORMATTED BINARY"
                        ; 376
                        ; 377   ; THE 4 LSB'S OF EACH 36-BIT OUTPUT WORD EQUAL ZERO.
                        ; 378
U 0111, 1145,4377,7060  ; 379   B.F32:  COND/SRC_ACK.XIP_1,J/BF32.X     ;wait for bytes 0 and 1
                        ; 380   =00
U 0114, 3657,4377,7060  ; 381   BF32.X: J/RX6X                          ;run out the Biker
U 0115, 1145,4377,7040  ; 382           COND/SRC_ACK.XIP_1,INREG_LOAD,J/BF32.X  ;NO ACK, XIP
U 0116, 4147,4377,7060  ; 383           J/READ.FAIL                             ;ACK, NO XIP
U 0117, 1207,4377,7040  ; 384           INREG_LOAD                      ;take bytes 0 and 1
                        ; 385   =
U 0120, 1217,5374,7460  ; 386           B15.D6                          ;get 2 bits
U 0121, 1227,5000,3460  ; 387           B13.D0,LOADX                    ;get 6 bits
U 0122, 1237,4374,7460  ; 388           B07.D6                          ;get 2 bits
U 0123, 1245,4000,5460  ; 389           B05.D0,LOADH,COND/SRC_ACK.XIP_1,J/BF321.X
                        ; 390                                           ;get 6 bits
                        ; 391                                           ;wait for bytes 2 and 3
                        ; 392   =00
U 0124, 4017,4377,7060  ; 393   BF321.X:J/RX6L                          ;finish byte with zeros
U 0125, 1245,4377,7040  ; 394           COND/SRC_ACK.XIP_1,INREG_LOAD,J/BF321.X ;NO ACK, XIP
U 0126, 4147,4377,7060  ; 395           J/READ.FAIL                             ;ACK, NO XIP
U 0127, 1307,4377,7040  ; 396           INREG_LOAD                      ;take bytes 2 and 3
                        ; 397   =
U 0130, 1327,5374,7460  ; 398           B15.D6                          ;get 2 bits
                        ; 399   =0
U 0132, 1327,0377,6060  ; 400   BF321.Z:LOADL,COND/OUTREG_MT,J/BF321.Z  ;wait until the Outreg is empty
U 0133, 1317,5774,7020  ; 401           B13.D6,OUTREG_LOAD              ;then load it with Word 0, Byte 0
                        ; 402                                           ;get 2 bits
                        ; 403   =
U 0131, 1347,4400,3060  ; 404           B11.D0,LOADX                    ;get 8 bits
U 0134, 1357,5400,5060  ; 405           B03.D0,LOADH                    ;get 4 bits
U 0135, 1367,6360,7060  ; 406           BZERO.D4                        ;get 4 bits of zero
                        ; 407   =0
U 0136, 1367,0377,6060  ; 408   BF322.Z:LOADL,COND/OUTREG_MT,J/BF322.Z  ;wait until the Outreg is empty
U 0137, 1117,4377,7020  ; 409           J/B.F32,OUTREG_LOAD             ;then load it with Word 0, Byte 1
                        ; 410   =
                        ; 411
```

```
                        ; 412   .TOC "MBC READ:  MODE 2      - 16-BIT BINARY TO 36-BIT BINARY"
                        ; 413
                        ; 414   ; NINE (9) 16-BIT WORDS ARE BASHED INTO FOUR (4) 36-BIT WORDS.
                        ; 415
U 0140, 1445,4377,7060  ; 416   B.B36:  COND/SRC_ACK.XIP_1,J/BB36.X     ;wait for Bytes 0 and 1
                        ; 417   =00
U 0144, 3657,4377,7060  ; 418   BB36.X: J/RX6X                          ;run out the Biker
U 0145, 1445,4377,7040  ; 419           COND/SRC_ACK.XIP_1,INREG_LOAD,J/BB36.X  ;NO ACK, XIP
U 0146, 4147,4377,7060  ; 420           J/READ.FAIL                             ;ACK, NO XIP
U 0147, 1417,4377,7040  ; 421           INREG_LOAD                      ;take bytes 0 and 1
                        ; 422   =
U 0141, 1427,5374,7460  ; 423           B15.D6                          ;get 2 bits
U 0142, 1437,5000,3460  ; 424           B13.D0,LOADX                    ;get 6 bits
U 0143, 1507,4374,7460  ; 425           B07.D6                          ;get 2 bits
U 0150, 1545,4000,5460  ; 426           B05.D0,LOADH,COND/SRC_ACK.XIP_1,J/BB361.X
                        ; 427                                           ;get 6 bits
                        ; 428                                           ;wait for Bytes 2 and 3
                        ; 429   =00
U 0154, 4017,4377,7060  ; 430   BB361.X:J/RX6L                          ;finish byte with zeros (same as F32 mode)
U 0155, 1545,4377,7040  ; 431           COND/SRC_ACK.XIP_1,INREG_LOAD,J/BB361.X ;NO ACK, XIP
U 0156, 4147,4377,7060  ; 432           J/READ.FAIL                             ;ACK, NO XIP
U 0157, 1517,4377,7040  ; 433           INREG_LOAD                      ;take bytes 0 and 1
                        ; 434   =
U 0151, 1527,5374,7460  ; 435           B15.D6                          ;get 2 bits
                        ; 436   =0
U 0152, 1527,0377,6060  ; 437   BB361.Z:LOADL,COND/OUTREG_MT,J/BB361.Z  ;wait until the Outreg is empty
U 0153, 1607,5774,7020  ; 438           B13.D6,OUTREG_LOAD              ;then load it with Word 0, Byte 0
                        ; 439                                           ;get 2 bits
                        ; 440   =
U 0160, 1617,4400,3060  ; 441           B11.D0,LOADX                    ;get 8 bits
U 0161, 1645,5400,5060  ; 442           B03.D0,LOADH,COND/SRC_ACK.XIP_1,J/BB362.X
                        ; 443                                           ;get 4 bits
                        ; 444                                           ;wait for Bytes 4 and 5
                        ; 445   =00
U 0164, 4007,4377,7060  ; 446   BB362.X:J/RX4L                          ;finish byte with zeros
U 0165, 1645,4377,7040  ; 447           COND/SRC_ACK.XIP_1,INREG_LOAD,J/BB362.X ;NO ACK, XIP
U 0166, 4147,4377,7060  ; 448           J/READ.FAIL                             ;ACK, NO XIP
U 0167, 1627,4377,7040  ; 449           INREG_LOAD                      ;take Bytes 4 and 5
                        ; 450   =
U 0162, 1707,5760,7060  ; 451           B15.D4                          ;get 4 bits
                        ; 452   =0
U 0170, 1707,0377,6060  ; 453   BB362.Z:LOADL,COND/OUTREG_MT,J/BB362.Z  ;wait until the Outreg is Empty
U 0171, 1637,4774,7420  ; 454           B11.D6,OUTREG_LOAD              ;then load it with Word 0, Byte 1
                        ; 455   =
U 0163, 1727,4400,3460  ; 456           B09.D0,LOADX                    ;get 6 bits
U 0172, 1737,5774,7460  ; 457           B03.D6                          ;get 2 bits
U 0173, 1745,5400,5460  ; 458           B01.D0,LOADH,COND/SRC_ACK.XIP_1,J/BB363.X
                        ; 459                                           ;get 2 bits
                        ; 460                                           ;wait for Bytes 6 and 7
                        ; 461   =00
U 0174, 3777,4377,7060  ; 462   BB363.X:J/RX2L                          ;finish byte with zeros
U 0175, 1745,4377,7040  ; 463           COND/SRC_ACK.XIP_1,INREG_LOAD,J/BB363.X ;NO ACK, XIP
U 0176, 4147,4377,7060  ; 464           J/READ.FAIL                             ;ACK, NO XIP
U 0177, 2007,4377,7040  ; 465           INREG_LOAD                      ;take Bytes 6 and 7
                        ; 466   =
U 0200, 2017,5700,7460  ; 467           B15.D2                          ;get 4 bits
U 0201, 2027,4774,7460  ; 468           B11.D6                          ;get 2 bits
                        ; 469   =0
U 0202, 2027,0377,6060  ; 470   BB363.Z:LOADL,COND/OUTREG_MT,J/BB363.Z  ;wait until the Outreg is empty
U 0203, 2047,5374,7020  ; 471           B09.D6,OUTREG_LOAD                      ;LOAD WORD 1, BYTE 0
                        ; 472   =
U 0204, 2105,4000,3060  ; 473           B07.D0,LOADX,COND/SRC_ACK.XIP_1,J/BB364.X       ;GET BYTE 8,9
                        ; 474   =00
U 0210, 3767,4377,5060  ; 475   BB364.X:LOADH,J/RX0L                          ;NO ACK, NO XIP
U 0211, 2105,4377,7040  ; 476           COND/SRC_ACK.XIP_1,INREG_LOAD,J/BB364.X ;NO ACK, XIP
U 0212, 4147,4377,7060  ; 477           J/READ.FAIL                             ;ACK, NO XIP
```

```
U 0213, 2057,4377,7040  ; 478          INREG_LOAD                                  ;ACK, XIP
                        ; 479  =
U 0205, 2067,5000,5060  ; 480          B15.D0,LOADH
                        ; 481  =0
U 0206, 2067,0377,6060  ; 482  BB364.Z:LOADL,COND/OUTREG_MT,J/BB364.Z
U 0207, 2147,4374,7420  ; 483          B07.D6,OUTREG_LOAD                          ;LOAD WORD 1, BYTE 1
                        ; 484  =
U 0214, 2205,4000,3460  ; 485          B05.D0,LOADX,COND/SRC_ACK.XIP_1,J/BB365.X    ;GET BYTES 10,11
                        ; 486  =00
U 0220, 3757,4377,7060  ; 487  BB365.X:J/RX6H                                      ;NO ACK, NO XIP
U 0221, 2205,4377,7040  ; 488          COND/SRC_ACK.XIP_1,INREG_LOAD,J/BB365.X      ;NO ACK, XIP
U 0222, 4147,4377,7060  ; 489          J/READ.FAIL                                 ;ACK, NO XIP
U 0223, 2157,4377,7040  ; 490          INREG_LOAD                                  ;ACK, XIP
                        ; 491  =
U 0215, 2167,5374,7460  ; 492          B15.D6
U 0216, 2177,5000,5460  ; 493          B13.D0,LOADH
U 0217, 2247,4374,7460  ; 494          B07.D6
                        ; 495  =0
U 0224, 2247,0377,6060  ; 496  BB365.Z:LOADL,COND/OUTREG_MT,J/BB365.Z
U 0225, 2267,4774,7020  ; 497          B05.D6,OUTREG_LOAD                          ;LOAD WORD 2, BYTE 0
                        ; 498  =
U 0226, 2305,5400,3060  ; 499          B03.D0,LOADX,COND/SRC_ACK.XIP_1,J/BB366.X    ;GET BYTES 12,13
                        ; 500  =00
U 0230, 3737,4377,7060  ; 501  BB366.X:J/RX4H                                      ;NO ACK, NO XIP
U 0231, 2305,4377,7040  ; 502          COND/SRC_ACK.XIP_1,INREG_LOAD,J/BB366.X      ;NO ACK, XIP
U 0232, 4147,4377,7060  ; 503          J/READ.FAIL                                 ;ACK, NO XIP
U 0233, 2277,4377,7040  ; 504          INREG_LOAD                                  ;ACK, XIP
                        ; 505  =
U 0227, 2347,5760,7060  ; 506          B15.D4
U 0234, 2367,4400,5060  ; 507          B11.D0,LOADH
                        ; 508  =0
U 0236, 2367,0377,6060  ; 509  BB366.Z:LOADL,COND/OUTREG_MT,J/BB366.Z
U 0237, 2357,5774,7420  ; 510          B03.D6,OUTREG_LOAD                          ;LOAD WORD 2, BYTE 1
                        ; 511  =
U 0235, 2405,5400,3460  ; 512          B01.D0,LOADX,COND/SRC_ACK.XIP_1,J/BB367.X    ;GET BYTES 14,15
                        ; 513  =00
U 0240, 3727,4377,7060  ; 514  BB367.X:J/RX2H                                      ;NO ACK, NO XIP
U 0241, 2405,4377,7040  ; 515          COND/SRC_ACK.XIP_1,INREG_LOAD,J/BB367.X      ;NO ACK, XIP
U 0242, 4147,4377,7060  ; 516          J/READ.FAIL                                 ;ACK, NO XIP
U 0243, 2447,4377,7040  ; 517          INREG_LOAD                                  ;ACK, XIP
                        ; 518  =
U 0244, 2457,5700,7460  ; 519          B15.D2
U 0245, 2467,4774,7460  ; 520          B11.D6
U 0246, 2477,4400,5460  ; 521          B09.D0,LOADH
U 0247, 2507,5774,7460  ; 522          B03.D6
                        ; 523  =0
U 0250, 2507,0377,6060  ; 524  BB367.Z:LOADL,COND/OUTREG_MT,J/BB367.Z
U 0251, 2527,4374,7020  ; 525          B01.D6,OUTREG_LOAD                          ;LOAD WORD 3, BYTE 0
                        ; 526  =
U 0252, 2545,4377,3060  ; 527          LOADX,COND/SRC_ACK.XIP_1,J/BB368.X           ;GET BYTES 16,17
                        ; 528  =00
U 0254, 3707,4377,7060  ; 529  BB368.X:J/RX0H                                      ;NO ACK, NO XIP
U 0255, 2545,4377,7040  ; 530          COND/SRC_ACK.XIP_1,INREG_LOAD,J/BB368.X      ;NO ACK, XIP
U 0256, 4147,4377,7060  ; 531          J/READ.FAIL                                 ;ACK, NO XIP
U 0257, 2537,4377,7040  ; 532          INREG_LOAD                                  ;ACK, XIP
                        ; 533  =
U 0253, 2607,5000,7060  ; 534          B15.D0
U 0260, 2627,4000,5060  ; 535          B07.D0,LOADH
                        ; 536  =0
U 0262, 2627,0377,6060  ; 537  BB369.Z:LOADL,COND/OUTREG_MT,J/BB369.Z
U 0263, 1407,4377,7020  ; 538          J/B.B36,OUTREG_LOAD                         ;LOAD WORD 3, BYTE 1
                        ; 539  =
                        ; 540
```

```
                    ; 541   .TOC "MBC READ:  MODE 3      - BYTE ASCII TO PACKED ASCII"
                    ; 542
                    ; 543   ; FIVE (5) 16-BIT WORDS ARE BASHED INTO TWO (2) 36-BIT WORDS
                    ; 544
U 0261, 2645,4377,7060  ; 545   BA.PA:  COND/SRC_ACK.XIP_1,J/BAPA.X             ;GET BYTES 0,1
                    ; 546   =00
U 0264, 3657,4377,7060  ; 547   BAPA.X: J/RX6X                                 ;NO ACK, NO XIP
U 0265, 2645,4377,7040  ; 548           COND/SRC_ACK.XIP_1,INREG_LOAD,J/BAPA.X  ;NO ACK, XIP
U 0266, 4147,4377,7060  ; 549           J/READ.FAIL                             ;ACK, NO XIP
U 0267, 2707,4377,7040  ; 550           INREG_LOAD                              ;ACK, XIP
                    ; 551   =
U 0270, 2717,5374,7660  ; 552           B14.D6
U 0271, 2727,5000,3660  ; 553           B12.D0,LOADX
U 0272, 2737,4770,7060  ; 554           B06.D5
U 0273, 2745,5400,5060  ; 555           B03.D0,LOADH,COND/SRC_ACK.XIP_1,J/BAPA1.X       ;GET BYTES 2,3
                    ; 556   =00
U 0274, 4007,4377,7060  ; 557   BAPA1.X:J/RX4L                                 ;NO ACK, NO XIP
U 0275, 2745,4377,7040  ; 558           COND/SRC_ACK.XIP_1,INREG_LOAD,J/BAPA1.X ;NO ACK, XIP
U 0276, 4147,4377,7060  ; 559           J/READ.FAIL                             ;ACK, NO XIP
U 0277, 3007,4377,7040  ; 560           INREG_LOAD                              ;ACK, XIP
                    ; 561   =
U 0300, 3017,5760,7260  ; 562           B14.D4
U 0301, 3027,4776,7260  ; 563           B11.D7
                    ; 564   =0
U 0302, 3027,0377,6060  ; 565   BAPA1.Z:LOADL,COND/OUTREG_MT,J/BAPA1.Z
U 0303, 3047,4774,7620  ; 566           B10.D6,OUTREG_LOAD                      ;LOAD WORD 0, BYTE 0
                    ; 567   =
U 0304, 3057,4400,3660  ; 568           B08.D0,LOADX
U 0305, 3105,4200,7060  ; 569           B06.D1,COND/SRC_ACK.XIP_1,J/BAPA2.X     ;GET BYTES 4,5
                    ; 570   =00
U 0310, 3767,4377,5060  ; 571   BAPA2.X:LOADH,J/RX0L                           ;NO ACK, NO XIP
U 0311, 3105,4377,7040  ; 572           COND/SRC_ACK.XIP_1,INREG_LOAD,J/BAPA2.X ;NO ACK, XIP
U 0312, 4147,4377,7060  ; 573           J/READ.FAIL                             ;ACK, NO XIP
U 0313, 3067,4377,7040  ; 574           INREG_LOAD                              ;ACK, XIP
                    ; 575   =
U 0306, 3077,5000,5260  ; 576           B14.D0,LOADH
U 0307, 3147,6376,7060  ; 577           BZERO.D7
                    ; 578   =0
U 0314, 3147,0377,6060  ; 579   BAPA2.Z:LOADL,COND/OUTREG_MT,J/BAPA2.Z
U 0315, 3167,4374,7620  ; 580           B06.D6,OUTREG_LOAD                      ;LOAD WORD 0, BYTE 1 (LSB=0)
                    ; 581   =
U 0316, 3205,4000,3660  ; 582           B04.D0,LOADX,COND/SRC_ACK.XIP_1,J/BAPA3.X       ;GET BYTES 6,7
                    ; 583   =00
U 0320, 3747,4377,7060  ; 584   BAPA3.X:J/RX5H                                 ;NO ACK, NO XIP
U 0321, 3205,4377,7040  ; 585           COND/SRC_ACK.XIP_1,INREG_LOAD,J/BAPA3.X ;NO ACK, XIP
U 0322, 4147,4377,7060  ; 586           J/READ.FAIL                             ;ACK, NO XIP
U 0323, 3177,4377,7040  ; 587           INREG_LOAD                              ;ACK, XIP
                    ; 588   =
U 0317, 3247,5770,7060  ; 589           B14.D5
U 0324, 3257,4400,5060  ; 590           B11.D0,LOADH
U 0325, 3267,4760,7260  ; 591           B06.D4
U 0326, 3307,5776,7260  ; 592           B03.D7
                    ; 593   =0
U 0330, 3307,0377,6060  ; 594   BAPA3.Z:LOADL,COND/OUTREG_MT,J/BAPA3.Z
U 0331, 3277,5774,7620  ; 595           B02.D6,OUTREG_LOAD                      ;LOAD WORD 1, BYTE 0
                    ; 596   =
U 0327, 3345,5400,3660  ; 597           B00.D0,LOADX,COND/SRC_ACK.XIP_1,J/BAPA4.X       ;GET BYTES 8,9
                    ; 598   =00
U 0334, 3717,4377,7060  ; 599   BAPA4.X:J/RX1H                                 ;NO ACK, NO XIP
U 0335, 3345,4377,7040  ; 600           COND/SRC_ACK.XIP_1,INREG_LOAD,J/BAPA4.X ;NO ACK, XIP
U 0336, 4147,4377,7060  ; 601           J/READ.FAIL                             ;ACK, NO XIP
U 0337, 3327,4377,7040  ; 602           INREG_LOAD                              ;ACK, XIP
                    ; 603   =
U 0332, 3337,5200,7060  ; 604           B14.D1
U 0333, 3407,4000,5260  ; 605           B06.D0,LOADH
U 0340, 3427,6376,7060  ; 606           BZERO.D7
```

```
                        ; 607   =0
U 0342, 3427,0377,6060  ; 608   BAPA4.Z:LOADL,COND/OUTREG_MT,J/BAPA4.Z
U 0343, 2617,4377,7020  ; 609           J/BA.PA,OUTREG_LOAD                     ;LOAD WORD 1, BYTE 1 (LSB=0)
                        ; 610   =
                        ; 611
```

```
                        ; 612   .TOC "MBC READ:  MODE 4      - 16-BIT BINARY TO 16-BIT F.B. WITH BYTE SWAP"
                        ; 613
                        ; 614   ; THE 2 MSB'S OF EACH 18-BIT OUTPUT BYTE EQUAL ZERO.
                        ; 615
U 0341, 3445,4377,7060  ; 616   B.S16:  COND/SRC_ACK.XIP_1,J/BS16.X             ;GET BYTES 0,1
                        ; 617   =00
U 0344, 3657,4377,7060  ; 618   BS16.X: J/RX6X                                 ;NO ACK, NO XIP
U 0345, 3445,4377,7040  ; 619           COND/SRC_ACK.XIP_1,INREG_LOAD,J/BS16.X  ;NO ACK, XIP
U 0346, 4147,4377,7060  ; 620           J/READ.FAIL                             ;ACK, NO XIP
U 0347, 3507,6374,7040  ; 621           BZERO.D6,INREG_LOAD                     ;ACK, XIP
                        ; 622   =
U 0350, 3517,4000,3060  ; 623           B07.D0,LOADX
U 0351, 3527,5000,5060  ; 624           B15.D0,LOADH
                        ; 625   =0
U 0352, 3527,0377,6060  ; 626   BS16.Z: LOADL,COND/OUTREG_MT,J/BS16.Z
U 0353, 3417,4377,7020  ; 627           OUTREG_LOAD,J/B.S16
                        ; 628   =
                        ; 629
```

```
                        ; 630   .TOC "MBC READ:  MODE 5      - 8-BIT BINARY TO 9-BIT BINARY"
                        ; 631
                        ; 632   ; THE MSB OF EACH 9-BIT OUTPUT BYTE EQUALS ZERO.
                        ; 633   ; NOTE THAT "INREG_LOAD" IS THE MIO OR MBUS REQUEST FOR MORE DATA !
                        ; 634
U 0354, 3605,4377,7060  ; 635   B.B09:  COND/SRC_ACK.XIP_1,J/BB09.X     ;wait for Bytes 0 and 1
                        ; 636   =00
U 0360, 3657,4377,7060  ; 637   BB09.X: J/RX6X                                  ;NO ACK, NO XIP
U 0361, 3605,4377,7040  ; 638          COND/SRC_ACK.XIP_1,INREG_LOAD,J/BB09.X  ;NO ACK, XIP
U 0362, 4147,4377,7060  ; 639          J/READ.FAIL                             ;ACK, NO XIP
U 0363, 3557,6374,7040  ; 640          BZERO.D6,INREG_LOAD                     ;ACK, XIP
                        ; 641   =                                       ;get 1 bit of zero
U 0355, 3567,5376,7260  ; 642          B15.D7                          ;get 1 bit
U 0356, 3577,5000,3260  ; 643          B14.D0,LOADX                    ;get 7 bits
U 0357, 3647,6376,7060  ; 644          BZERO.D7                        ;get 1 bit of zero
U 0364, 3667,4000,5060  ; 645          B07.D0,LOADH                    ;get 8 bits
                        ; 646   =0
U 0366, 3667,0377,6060  ; 647   BB09.Z: LOADL,COND/OUTREG_MT,J/BB09.Z   ;wait until the Outreg is empty
U 0367, 3547,4377,7020  ; 648          OUTREG_LOAD,J/B.B09             ;then load it with Word 0, Byte 0
                        ; 649   ==
                        ; 650
```

```
                        ; 651   .TOC "MBC READ:  END CODE    - ALL READ MODES"
                        ; 652
                        ; 653   ; THESE ARE THE END ROUTINES FOR ALL OF THE READ MODES.
                        ; 654
                        ; 655   ; The correct end routine is found by examining the next Bxx.Dn and LOADz
                        ; 656   ; following it, where "n" ranges from 0 to 7 and "z" is one of "X", "H", or "L".
                        ; 657   ; The end routine is then RXnz.
                        ; 658
                        ; 659   ; **** Enter here to finish the byte with the correct number of zero bits:
                        ; 660
U 0365, 4037,6374,7060  ; 661   RX6X:   BZERO.D6,J/RX.X                ;get 2 bits of zero and go finish up
                        ; 662
                        ; 663
U 0370, 4047,6000,7060  ; 664   RX0H:   BZERO.D0,J/RX.H                ;get 8 bits of zero and go finish up
                        ; 665
U 0371, 4047,6200,7060  ; 666   RX1H:   BZERO.D1,J/RX.H                ;get 7 bits of zero and go finish up
                        ; 667
U 0372, 4047,6300,7060  ; 668   RX2H:   BZERO.D2,J/RX.H                ;get 6 bits of zero and go finish up
                        ; 669
U 0373, 4047,6360,7060  ; 670   RX4H:   BZERO.D4,J/RX.H                ;get 4 bits of zero and go finish up
                        ; 671
U 0374, 4047,6370,7060  ; 672   RX5H:   BZERO.D5,J/RX.H                ;get 3 bits of zero and go finish up
                        ; 673
U 0375, 4047,6374,7060  ; 674   RX6H:   BZERO.D6,J/RX.H                ;get 2 bits of zero and go finish up
                        ; 675
                        ; 676
U 0376, 4057,6000,7060  ; 677   RX0L:   BZERO.D0,J/RX.L                ;get 8 bits of zero and go finish up
                        ; 678
U 0377, 4057,6300,7060  ; 679   RX2L:   BZERO.D2,J/RX.L                ;get 6 bits of zero and go finish up
                        ; 680
U 0400, 4057,6360,7060  ; 681   RX4L:   BZERO.D4,J/RX.L                ;get 4 bits of zero and go finish up
                        ; 682
U 0401, 4057,6374,7060  ; 683   RX6L:   BZERO.D6,J/RX.L                ;get 2 bits of zero and go finish up
                        ; 684
                        ; 685   ; **** This loop generates zero-data SCLKS until the Biker is zero
                        ; 686
U 0402, 4037,6374,7060  ; 687   RX:     BZERO.D6                       ;get 2 bits of zero
U 0403, 4047,6000,3060  ; 688   RX.X:   BZERO.D0,LOADX                 ;load X-byte and get 8 bits of zero for H
U 0404, 4057,6000,5060  ; 689   RX.H:   BZERO.D0,LOADH                 ;load H-byte and get 8 bits of zero for L
U 0405, 4067,0377,6060  ; 690   RX.L:   LOADL,COND/OUTREG_MT,J/RX.L.X   ;load L-byte and wait for Outreg Empty
                        ; 691   =0
U 0406, 4067,0377,7060  ; 692   RX.L.X: COND/OUTREG_MT,J/RX.L.X         ;wait until the Outreg is empty
U 0407, 4105,0377,7060  ; 693           COND/SRC_ACK.XIP_0,J/RX.B.X     ;see if the Biker is zero
                        ; 694   =
                        ; 695   =00
U 0410, 0007,4377,7160  ; 696   RX.B.X: END_XFER,J/IDLE                 ;Biker is zero, get out
U 0411, 4027,4377,7020  ; 697           J/RX,OUTREG_LOAD                ;Biker is not zero, continue
                        ; 698                                           ;load the Outreg with a zero-data byte
U 0412, 4147,4377,7060  ; 699           J/READ.FAIL                     ;ACK, NO XIP
U 0413, 4027,4377,7020  ; 700           J/RX,OUTREG_LOAD                ;Biker is not zero, continue
                        ; 701                                           ;load the Outreg with a zero-data byte
                        ; 702   =
                        ; 703
                        ; 704   ;**** COMMON FAIL CODE FOR LOGIC ANALYZER TRIGGER ****
U 0414, 0007,4377,7160  ; 705   READ.FAIL:      END_XFER,J/IDLE
                        ; 706
                        ; 707
```

```
                        ; 708   .TOC "MBC WRITE: HEADER MODE 0 - 16-BIT FORMATTED BINARY TO 16-BIT BINARY"
                        ; 709
                        ; 710   ; THE 2 MSB'S OF EACH 18-BIT INPUT BYTE ARE ZERO.
                        ; 711
                        ; 712   =00
U 0420, 7547,4377,7060  ; 713   HW16.X: J/WRT.END                                  ;END XFER, ALL DONE
U 0421, 4205,0377,7040  ; 714           COND/SRC_ACK.XIP_0,INREG_LOAD,J/HW16.X   ;NO ACK, XIP
U 0422, 7567,4377,7060  ; 715           J/WRITE.FAIL                             ;ACK, NO XIP
U 0423, 4157,4377,7040  ; 716           INREG_LOAD                               ;ACK, XIP
                        ; 717   =:
U 0415, 4167,5000,7060  ; 718           BX02.D0
U 0416, 4247,4000,5060  ; 719           BX10.D0,LOADH
                        ; 720   =0
U 0424, 4247,0377,6060  ; 721   HW16.Z: LOADL,COND/OUTREG_MT,J/HW16.Z
U 0425, 4177,4377,7020  ; 722           OUTREG_LOAD                              ;LOAD WORD 0 AND DECREMENT BIKER
                        ; 723   =:
U 0417, 4305,0377,7060  ; 724           COND/SRC_ACK.XIP_0,J/HW161.X              ;TEST BIKER
                        ; 725   =00
U 0430, 7547,4377,7060  ; 726   HW161.X:J/WRT.END                                ;END XFER, 1 OR 2 BYTES OUT
U 0431, 4305,0377,7040  ; 727           COND/SRC_ACK.XIP_0,INREG_LOAD,J/HW161.X  ;NO ACK, XIP
U 0432, 7567,4377,7060  ; 728           J/WRITE.FAIL                             ;ACK, NO XIP
U 0433, 4267,4377,7040  ; 729           INREG_LOAD                               ;ACK, XIP
                        ; 730   =:
U 0426, 4277,5000,7060  ; 731           BX02.D0
U 0427, 4347,4000,5060  ; 732           BX10.D0,LOADH
                        ; 733   =0
U 0434, 4347,0377,6060  ; 734   HW161.Z:LOADL,COND/OUTREG_MT,J/HW161.Z
U 0435, 4367,4377,7020  ; 735           OUTREG_LOAD                              ;LOAD WORD 1 AND DECREMENT BIKER
                        ; 736   =:
U 0436, 4406,0377,7060  ; 737           COND/HDR_MODE,J/HW16.HDR.X                ;SEE IF WE ARE STILL IN HEADER MODE
                        ; 738   =0
U 0440, 0200,0377,7060  ; 739   HW16.HDR.X: COND/MODE,J/MW.X                      ;END OF HEADER MODE
U 0441, 4205,0377,7060  ; 740           COND/SRC_ACK.XIP_0,J/HW16.X               ;CONTINUE IN HEADER MODE
                        ; 741   =:
                        ; 742
```

```
                        ; 743   .TOC "MBC WRITE: HEADER MODE 1 - 32-BIT FORMATTED BINARY TO 16-BIT BINARY"
                        ; 744
                        ; 745   ; THE 4 LSB'S OF EACH 36-BIT INPUT WORD ARE IGNORED.
                        ; 746
                        ; 747   =00                                          ;GET WORD 0, BYTE 0
U 0444, 7547,4377,7060  ; 748   HW32.X: J/WRT.END                            ;FULL CYCLE OUT, ALL DONE
U 0445, 4445,0377,7040  ; 749           COND/SRC_ACK.XIP_0,INREG_LOAD,J/HW32.X  ;NO ACK, XIP
U 0446, 7567,4377,7060  ; 750           J/WRITE.FAIL                         ;ACK, NO XIP
U 0447, 4377,4377,7040  ; 751           INREG_LOAD                           ;ACK, XIP
                        ; 752   =
U 0437, 4427,7400,7460  ; 753           BX00.D0
U 0442, 4437,4774,7460  ; 754           BX06.D6
U 0443, 4507,4400,5460  ; 755           BX08.D0,LOADH
U 0450, 4527,5774,7460  ; 756           BX14.D6
                        ; 757   =0
U 0452, 4527,0377,6060  ; 758   HW32.Z: LOADL,COND/OUTREG_MT,J/HW32.Z
U 0453, 4517,4377,7020  ; 759           OUTREG_LOAD                          ;LOAD WORD 0
                        ; 760   =
U 0451, 4545,1400,7460  ; 761           BX16.D0,COND/SRC_ACK.XIP_0,J/HW321.X
                        ; 762   =00                                          ;GET WORD 0, BYTE 1
U 0454, 7547,4377,7060  ; 763   HW321.X:J/WRT.END                           ;1 OR 2 BYTES OUT
U 0455, 4545,0377,7040  ; 764           COND/SRC_ACK.XIP_0,INREG_LOAD,J/HW321.X ;NO ACK, XIP
U 0456, 7567,4377,7060  ; 765           J/WRITE.FAIL                         ;ACK, NO XIP
U 0457, 4607,4377,7040  ; 766           INREG_LOAD                           ;ACK, XIP
                        ; 767   =
U 0460, 4617,7700,7060  ; 768           BX18.D2
U 0461, 4627,4400,5060  ; 769           BX24.D0,LOADH
                        ; 770   =0
U 0462, 4627,0377,6060  ; 771   HW321.Z:LOADL,COND/OUTREG_MT,J/HW321.Z
U 0463, 4647,4377,7020  ; 772           OUTREG_LOAD                          ;LOAD WORD 1
                        ; 773   =                                            ;3 OR 4 BYTES OUT
U 0464, 4666,0377,7060  ; 774           COND/HDR_MODE,J/HW32.HDR.X           ;SEE IF WE ARE STILL IN HEADER MODE
                        ; 775   =0
U 0466, 0200,0377,7060  ; 776   HW32.HDR.X: COND/MODE,J/MW.X                  ;END OF HEADER MODE
U 0467, 4445,0377,7060  ; 777           COND/SRC_ACK.XIP_0,J/HW32.X           ;CONTINUE IN HEADER MODE
                        ; 778   =
                        ; 779
```

```
                        ; 780   .TOC "MBC WRITE: MODE 0      - 16-BIT FORMATTED BINARY TO 16-BIT BINARY"
                        ; 781
                        ; 782   ; THE 2 MSB'S OF EACH 18-BIT INPUT BYTE ARE IGNORED.
                        ; 783
U 0465, 4705,0377,7060  ; 784   F16.B:  COND/SRC_ACK.XIP_0,J/F16B.X             ;GET BYTE 0
                        ; 785   =00
U 0470, 7547,4377,7060  ; 786   F16B.X: J/WRT.END                               ;FULL CYCLE OUT, ALL DONE
U 0471, 4705,0377,7040  ; 787           COND/SRC_ACK.XIP_0,INREG_LOAD,J/F16B.X  ;NO ACK, XIP
U 0472, 7567,4377,7060  ; 788           J/WRITE.FAIL                            ;ACK, NO XIP
U 0473, 4747,4377,7040  ; 789           INREG_LOAD                              ;ACK, XIP
                        ; 790   =
U 0474, 4757,5000,7060  ; 791           BX02.D0
U 0475, 4767,4000,5060  ; 792           BX10.D0,LOADH
                        ; 793   =0
U 0476, 4767,0377,6060  ; 794   F16B.Z: LOADL,COND/OUTREG_MT,J/F16B.Z
U 0477, 4657,4377,7020  ; 795           OUTREG_LOAD,J/F16.B
                        ; 796   =
                        ; 797
```

```
                        ; 798   .TOC "MBC WRITE: MODE 1      - 32-BIT FORMATTED BINARY TO 16-BIT BINARY"
                        ; 799
                        ; 800   ; THE 4 LSB'S OF EACH 36-BIT INPUT WORD ARE IGNORED.
                        ; 801
U 0500, 5045,0377,7060  ; 802   F32.B:  COND/SRC_ACK.XIP_0,J/F32B.X             ;GET WORD 0, BYTE 0
                        ; 803   =00
U 0504, 7547,4377,7060  ; 804   F32B.X: J/WRT.END                               ;FULL CYCLE OUT, ALL DONE
U 0505, 5045,0377,7040  ; 805           COND/SRC_ACK.XIP_0,INREG_LOAD,J/F32B.X  ;NO ACK, XIP
U 0506, 7567,4377,7060  ; 806           J/WRITE.FAIL                            ;ACK, NO XIP
U 0507, 5017,4377,7040  ; 807           INREG_LOAD                              ;ACK, XIP
                        ; 808   =
U 0501, 5027,7400,7460  ; 809           BX00.D0
U 0502, 5037,4774,7460  ; 810           BX06.D6
U 0503, 5107,4400,5460  ; 811           BX08.D0,LOADH
U 0510, 5127,5774,7460  ; 812           BX14.D6
                        ; 813   =0
U 0512, 5127,0377,6060  ; 814   F32B.Z: LOADL,COND/OUTREG_MT,J/F32B.Z
U 0513, 5117,4377,7020  ; 815           OUTREG_LOAD
                        ; 816   =
U 0511, 5145,1400,7460  ; 817           BX16.D0,COND/SRC_ACK.XIP_0,J/F32B1.X     ;LOAD WORD 0
                        ; 818   =00                                             ;GET WORD 0, BYTE 1
U 0514, 7547,4377,7060  ; 819   F32B1.X:J/WRT.END                               ;1 OR 2 BYTES OUT
U 0515, 5145,0377,7040  ; 820           COND/SRC_ACK.XIP_0,INREG_LOAD,J/F32B1.X ;NO ACK, XIP
U 0516, 7567,4377,7060  ; 821           J/WRITE.FAIL                            ;ACK, NO XIP
U 0517, 5207,4377,7040  ; 822           INREG_LOAD                              ;ACK, XIP
                        ; 823   =
U 0520, 5217,7700,7060  ; 824           BX18.D2
U 0521, 5227,4400,5060  ; 825           BX24.D0,LOADH
                        ; 826   =0
U 0522, 5227,0377,6060  ; 827   F32B1.Z:LOADL,COND/OUTREG_MT,J/F32B1.Z
U 0523, 5247,4377,7020  ; 828           OUTREG_LOAD
                        ; 829   =
U 0524, 5045,0377,7060  ; 830           COND/SRC_ACK.XIP_0,J/F32B.X              ;LOAD WORD 1
                        ; 831                                                   ;3 OR 4 BYTES OUT
                        ; 832
                        ; 833
```

```
                        ; 834   .TOC "MBC WRITE: MODE 2      - 36-BIT BINARY TO 16-BIT BINARY"
                        ; 835
                        ; 836   ; FOUR (4) 36-BIT WORDS ARE BASHED INTO NINE (9) 16-BIT WORDS.
                        ; 837
U 0525, 5305,4377,7060  ; 838   B36.B:  COND/SRC_ACK.XIP_1,J/B36B.X             ;GET WORD 0, BYTE 0
                        ; 839   =00
U 0530, 7547,4377,7060  ; 840   B36B.X: J/WRT.END                               ;FULL CYCLE OUT, ALL DONE
U 0531, 5305,4377,7040  ; 841           COND/SRC_ACK.XIP_1,INREG_LOAD,J/B36B.X  ;NO ACK, XIP
U 0532, 7567,4377,7060  ; 842           J/WRITE.FAIL                            ;ACK, NO XIP
U 0533, 5267,4377,7040  ; 843           INREG_LOAD                              ;ACK, XIP
                        ; 844   =
U 0526, 5277,7400,7460  ; 845           BX00.D0                 ;get 6 bits
U 0527, 5347,4774,7460  ; 846           BX06.D6                 ;get 2 bits
U 0534, 5357,4400,5460  ; 847           BX08.D0,LOADH           ;load. get 6 bits
U 0535, 5367,5774,7460  ; 848           BX14.D6                 ;get 2 bits
                        ; 849   =0
U 0536, 5367,0377,6060  ; 850   B36B.Z: LOADL,COND/OUTREG_MT,J/B36B.Z            ;WAIT FOR OUTREG EMPTY
U 0537, 5407,4377,7020  ; 851           OUTREG_LOAD                             ;LOAD WORD 0
                        ; 852   =
U 0540, 5445,5400,7460  ; 853           BX16.D0,COND/SRC_ACK.XIP_1,J/B36B1.X    ;BRANCH ON INPUT READY
                        ; 854                                   ;get 2 bits
                        ; 855   =00                                             ;GET WORD 0, BYTE 1
U 0544, 7547,4377,7060  ; 856   B36B1.X:J/WRT.END                               ;1 OR 2 BYTES OUT
U 0545, 5445,4377,7040  ; 857           COND/SRC_ACK.XIP_1,INREG_LOAD,J/B36B1.X ;NO ACK, XIP
U 0546, 7567,4377,7060  ; 858           J/WRITE.FAIL                            ;ACK, NO XIP
U 0547, 5417,4377,7040  ; 859           INREG_LOAD                              ;ACK, XIP
                        ; 860   =
U 0541, 5427,7700,7060  ; 861           BX18.D2                 ;get 6 bits
U 0542, 5507,4400,5060  ; 862           BX24.D0,LOADH           ;load. get 8 bits
                        ; 863   =0
U 0550, 5507,0377,6060  ; 864   B36B1.Z:LOADL,COND/OUTREG_MT,J/B36B1.Z          ;WAIT FOR OUTREG EMPTY
U 0551, 5437,4377,7020  ; 865           OUTREG_LOAD                             ;LOAD WORD 1
                        ; 866   =
U 0543, 5545,5400,7060  ; 867           BX32.D0,COND/SRC_ACK.XIP_1,J/B36B2.X    ;BRANCH ON INPUT READY
                        ; 868                                   ;get 4 bits
                        ; 869   =00                                             ;GET WORD 1, BYTE 0
U 0554, 7507,4377,5060  ; 870   B36B2.X:LOADH,J/WE.1B36                         ;1 36-BIT WORD OUT, OUTPUT 12 BITS
U 0555, 5545,4377,7040  ; 871           COND/SRC_ACK.XIP_1,INREG_LOAD,J/B36B2.X ;NO ACK, XIP
U 0556, 7567,4377,7060  ; 872           J/WRITE.FAIL                            ;ACK, NO XIP
U 0557, 5527,4377,7040  ; 873           INREG_LOAD                              ;ACK, XIP
                        ; 874   =
U 0552, 5537,7360,7460  ; 875           BX00.D4                 ;get 4 bits
U 0553, 5607,5000,5460  ; 876           BX04.D0,LOADH           ;load. get 6 bits
U 0560, 5627,4374,7460  ; 877           BX10.D6                 ;get 2 bits
                        ; 878   =0
U 0562, 5627,0377,6060  ; 879   B36B2.Z:LOADL,COND/OUTREG_MT,J/B36B2.Z          ;WAIT FOR OUTREG EMPTY
U 0563, 5617,4000,7420  ; 880           BX12.D0,OUTREG_LOAD     ;get 6 bits     ;LOAD WORD 2
                        ; 881   =
U 0561, 5645,4377,7060  ; 882           COND/SRC_ACK.XIP_1,J/B36B3.X            ;GET WORD 1, BYTE 1
                        ; 883   =00
U 0564, 7547,4377,7060  ; 884   B36B3.X:J/WRT.END                               ;5 OR 6 BYTES OUT
U 0565, 5645,4377,7040  ; 885           COND/SRC_ACK.XIP_1,INREG_LOAD,J/B36B3.X ;NO ACK, XIP
U 0566, 7567,4377,7060  ; 886           J/WRITE.FAIL                            ;ACK, NO XIP
U 0567, 5707,4377,7040  ; 887           INREG_LOAD                              ;ACK, XIP
                        ; 888   =
U 0570, 5717,7374,7060  ; 889           BX18.D6                 ;get 2 bits
U 0571, 5727,5000,5060  ; 890           BX20.D0,LOADH           ;load. get 8 bits
                        ; 891   =0
U 0572, 5727,0377,6060  ; 892   B36B3.Z:LOADL,COND/OUTREG_MT,J/B36B3.Z          ;WAIT FOR OUTREG EMPTY
U 0573, 5747,4000,7020  ; 893           BX28.D0,OUTREG_LOAD     ;get 8 bits     ;LOAD WORD 3
                        ; 894   =
U 0574, 6005,4377,5060  ; 895           LOADH,COND/SRC_ACK.XIP_1,J/B36B4.X      ;GET WORD 2, BYTE 0
                        ; 896   =00
U 0600, 7477,4377,7060  ; 897   B36B4.X:J/WRT.END.O                             ;2 36-BIT WORDS OUT, OUTPUT 8 BITS
U 0601, 6005,4377,7040  ; 898           COND/SRC_ACK.XIP_1,INREG_LOAD,J/B36B4.X ;NO ACK, XIP
U 0602, 7567,4377,7060  ; 899           J/WRITE.FAIL                            ;ACK, NO XIP
```

```
U 0603, 5757,4377,7040  ; 900          INREG_LOAD                                ;ACK, XIP
                        ; 901  =
U 0575, 5767,7400,7460  ; 902          BX00.D0                ;get 6 bits
U 0576, 6047,4774,7460  ; 903          BX06.D6                ;get 2 bits
                        ; 904  =0
U 0604, 6047,0377,6060  ; 905  B36B4.Z:LOADL,COND/OUTREG_MT,J/B36B4.Z
U 0605, 5777,4400,7420  ; 906          BX08.D0,OUTREG_LOAD    ;get 6 bits     ;LOAD WORD 4
                        ; 907  =
U 0577, 6067,5774,7460  ; 908          BX14.D6                ;get 2 bits
U 0606, 6105,5400,5460  ; 909          BX16.D0,LOADH,COND/SRC_ACK.XIP_1,J/B36B5.X      ;GET WORD 2, BYTE 1
                        ; 910                                 ;get 2 bits
                        ; 911  =00
U 0610, 7477,4377,7060  ; 912  B36B5.X:J/WRT.END.O                               ;10 OR 11 BYTES OUT, NEED OUTPUT
U 0611, 6105,4377,7040  ; 913          COND/SRC_ACK.XIP_1,INREG_LOAD,J/B36B5.X ;NO ACK, XIP
U 0612, 7567,4377,7060  ; 914          J/WRITE.FAIL                              ;ACK, NO XIP
U 0613, 6077,4377,7040  ; 915          INREG_LOAD                                ;ACK, XIP
                        ; 916  =
U 0607, 6147,7700,7060  ; 917          BX18.D2                ;get 6 bits
                        ; 918  =0
U 0614, 6147,0377,6060  ; 919  B36B5.Z:LOADL,COND/OUTREG_MT,J/B36B5.Z
U 0615, 6167,4400,7020  ; 920          BX24.D0,OUTREG_LOAD    ;get 8 bits     ;LOAD WORD 5
                        ; 921  =
U 0616, 6205,5400,5060  ; 922          BX32.D0,LOADH,COND/SRC_ACK.XIP_1,J/B36B6.X      ;GET WORD 3, BYTE 0
                        ; 923                                 ;get 4 bits
                        ; 924  =00
U 0620, 7557,4377,6060  ; 925  B36B6.X:LOADL,J/WE.2B36                           ;3 36-BIT WORDS OUT, OUTPUT 4 BITS
U 0621, 6205,4377,7040  ; 926          COND/SRC_ACK.XIP_1,INREG_LOAD,J/B36B6.X ;NO ACK, XIP
U 0622, 7567,4377,7060  ; 927          J/WRITE.FAIL                              ;ACK, NO XIP
U 0623, 6177,4377,7040  ; 928          INREG_LOAD                                ;ACK, XIP
                        ; 929  =
U 0617, 6247,7360,7460  ; 930          BX00.D4                ;get 4 bits
                        ; 931  =0
U 0624, 6247,0377,6060  ; 932  B36B6.Z:LOADL,COND/OUTREG_MT,J/B36B6.Z
U 0625, 6267,5000,7420  ; 933          BX04.D0,OUTREG_LOAD    ;get 6 bits     ;LOAD WORD 6
                        ; 934  =
U 0626, 6277,4374,7460  ; 935          BX10.D6                ;get 2 bits
U 0627, 6305,4000,5460  ; 936          BX12.D0,LOADH,COND/SRC_ACK.XIP_1,J/B36B7.X      ;GET WORD 3, BYTE 1
                        ; 937                                 ;get 6 bits
                        ; 938  =00
U 0630, 7477,4377,7060  ; 939  B36B7.X:J/WRT.END.O                               ;14 OR 15 BYTES OUT, NEED OUTPUT
U 0631, 6305,4377,7040  ; 940          COND/SRC_ACK.XIP_1,INREG_LOAD,J/B36B7.X ;NO ACK, XIP
U 0632, 7567,4377,7060  ; 941          J/WRITE.FAIL                              ;ACK, NO XIP
U 0633, 6347,4377,7040  ; 942          INREG_LOAD                                ;ACK, XIP
                        ; 943  =
U 0634, 6367,7374,7060  ; 944          BX18.D6                ;get 2 bits
                        ; 945  =0
U 0636, 6367,0377,6060  ; 946  B36B7.Z:LOADL,COND/OUTREG_MT,J/B36B7.Z
U 0637, 6357,5000,7020  ; 947          BX20.D0,OUTREG_LOAD    ;get 8 bits     ;LOAD WORD 7
                        ; 948  =                                               ;16 BYTES OUT
U 0635, 6407,4000,5060  ; 949          BX28.D0,LOADH          ;get 8 bits
                        ; 950  =0
U 0640, 6407,0377,6060  ; 951  B36B8.Z:LOADL,COND/OUTREG_MT,J/B36B8.Z
U 0641, 5257,4377,7020  ; 952          J/B36.B,OUTREG_LOAD                       ;LOAD WORD 8
                        ; 953  =                                               ;4 36-BIT WORDS OUT
                        ; 954
                        ; 955
```

```
                        ; 956   .TOC "MBC WRITE: MODE 3      - PACKED ASCII TO BYTE ASCII"
                        ; 957
                        ; 958   ; TWO (2) 36-BIT WORDS ARE BASHED INTO FIVE (5) 16-BIT WORDS.
                        ; 959
U 0642, 6445,0377,7060  ; 960   PA.BA:  COND/SRC_ACK.XIP_0,J/PABA.X              ;GET WORD 0, BYTE 0
                        ; 961   =00
U 0644, 7547,4377,7060  ; 962   PABA.X: J/WRT.END                               ;FULL CYCLE OUT, ALL DONE
U 0645, 6445,0377,7040  ; 963           COND/SRC_ACK.XIP_0,INREG_LOAD,J/PABA.X  ;NO ACK, XIP
U 0646, 7567,4377,7060  ; 964           J/WRITE.FAIL                            ;ACK, NO XIP
U 0647, 6437,6000,7040  ; 965           BZERO.D0,INREG_LOAD                     ;ACK, XIP
                        ; 966   =
U 0643, 6507,7600,7260  ; 967           BX00.D1
U 0650, 6517,4776,7260  ; 968           BX06.D7
U 0651, 6527,4600,5060  ; 969           BX07.D1,LOADH
                        ; 970   =0
U 0652, 6527,0377,6060  ; 971   PABA.Z: LOADL,COND/OUTREG_MT,J/PABA.Z
U 0653, 6547,4377,7020  ; 972           OUTREG_LOAD
                        ; 973   =
U 0654, 6605,0200,7660  ; 974           BX14.D1,COND/SRC_ACK.XIP_0,J/PABA1.X     ;LOAD WORD 0
                        ; 975   =00                                             ;GET WORD 0, BYTE 1
U 0660, 7547,4377,7060  ; 976   PABA1.X:J/WRT.END                               ;1 OR 2 BYTES OUT
U 0661, 6605,0377,7040  ; 977           COND/SRC_ACK.XIP_0,INREG_LOAD,J/PABA1.X ;NO ACK, XIP
U 0662, 7567,4377,7060  ; 978           J/WRITE.FAIL                            ;ACK, NO XIP
U 0663, 6557,4377,7040  ; 979           INREG_LOAD                              ;ACK, XIP
                        ; 980   =
U 0655, 6567,7370,7260  ; 981           BX18.D5
U 0656, 6647,5200,5060  ; 982           BX21.D1,LOADH
                        ; 983   =0
U 0664, 6647,0377,6060  ; 984   PABA1.Z:LOADL,COND/OUTREG_MT,J/PABA1.Z
U 0665, 6577,4600,7620  ; 985           BX28.D1,OUTREG_LOAD                     ;LOAD WORD 1
                        ; 986   =
U 0657, 6705,5770,7660  ; 987           BX32.D5,COND/SRC_ACK.XIP_1,J/PABA2.X     ;GET WORD 1, BYTE 0
                        ; 988   =00
U 0670, 7445,0377,5060  ; 989   PABA2.X:COND/SRC_ACK.XIP_0,J/PABA.EXIT,LOADH     ;3,4 OR 5 BYTES OUT, TEST WHICH
U 0671, 6705,4377,7040  ; 990           COND/SRC_ACK.XIP_1,INREG_LOAD,J/PABA2.X ;NO ACK, XIP
U 0672, 7567,4377,7060  ; 991           J/WRITE.FAIL                            ;ACK, NO XIP
U 0673, 6667,4377,7040  ; 992           INREG_LOAD                              ;ACK, XIP
                        ; 993   =
U 0666, 6677,7600,5260  ; 994           BX00.D1,LOADH
U 0667, 6747,4776,7260  ; 995           BX06.D7
                        ; 996   =0
U 0674, 6747,0377,6060  ; 997   PABA2.Z:LOADL,COND/OUTREG_MT,J/PABA2.Z
U 0675, 6767,4600,7020  ; 998           BX07.D1,OUTREG_LOAD                     ;LOAD WORD 2
                        ; 999   =
U 0676, 7005,0200,5660  ; 1000          BX14.D1,LOADH,COND/SRC_ACK.XIP_0,J/PABA3.X       ;GET WORD 1, BYTE 1
                        ; 1001  =00
U 0700, 7547,4377,7060  ; 1002  PABA3.X:J/WRT.END                               ;6 BYTES OUT, EXIT
U 0701, 7005,0377,7040  ; 1003          COND/SRC_ACK.XIP_0,INREG_LOAD,J/PABA3.X ;NO ACK, XIP
U 0702, 7567,4377,7060  ; 1004          J/WRITE.FAIL                            ;ACK, NO XIP
U 0703, 6777,4377,7040  ; 1005          INREG_LOAD                              ;ACK, XIP
                        ; 1006  =
U 0677, 7047,7370,7260  ; 1007          BX18.D5
                        ; 1008  =0
U 0704, 7047,0377,6060  ; 1009  PABA3.Z:LOADL,COND/OUTREG_MT,J/PABA3.Z
U 0705, 7067,5200,7020  ; 1010          BX21.D1,OUTREG_LOAD                     ;LOAD WORD 3
                        ; 1011  =                                               ;THIS IS FOR 8 BYTES OUT
U 0706, 7105,0377,7060  ; 1012          COND/SRC_ACK.XIP_0      ;WE ARE ONLY INTERESTED IN TESTING THE BYTE COUNT
                        ; 1013  =00
U 0710, 7547,4377,7060  ; 1014  PABA4.X:J/WRT.END                               ;7 OR 8 BYTES OUT, EXIT
U 0711, 7137,4377,7060  ; 1015          J/PABA4.Y
U 0712, 7547,4377,7060  ; 1016          J/WRT.END                               ;7 OR 8 BYTES OUT, EXIT
U 0713, 7077,4600,5660  ; 1017  PABA4.Y:BX28.D1,LOADH
                        ; 1018  =
U 0707, 7147,5770,7660  ; 1019          BX32.D5
                        ; 1020  =0
U 0714, 7147,0377,6060  ; 1021  PABA4.Z:LOADL,COND/OUTREG_MT,J/PABA4.Z
```

```
U 0715, 6427,4377,7020  ; 1022          J/PA.BA,OUTREG_LOAD                 ;LOAD WORD 4
                        ; 1023  =                                           ;THIS IS FOR 9 OR 10 BYTES OUT
                        ; 1024
                        ; 1025
```

```
                        ; 1026  .TOC "MBC WRITE: MODE 4      - 16-BIT F.B. TO 16-BIT BINARY WITH BYTE SWAP"
                        ; 1027
                        ; 1028  ; THE 2 MSB'S OF EACH 18-BIT INPUT BYTE ARE IGNORED.
                        ; 1029
U 0716, 7205,0377,7060  ; 1030  F16.S:  COND/SRC_ACK.XIP_0,J/F16S.X             ;GET BYTE 0
                        ; 1031  =00
U 0720, 7547,4377,7060  ; 1032  F16S.X: J/WRT.END                               ;FULL CYCLE OUT, ALL DONE
U 0721, 7205,0377,7040  ; 1033          COND/SRC_ACK.XIP_0,INREG_LOAD,J/F16S.X  ;NO ACK, XIP
U 0722, 7567,4377,7060  ; 1034          J/WRITE.FAIL                            ;ACK, NO XIP
U 0723, 7177,4377,7040  ; 1035          INREG_LOAD                              ;ACK, XIP
                        ; 1036  =
U 0717, 7247,4000,7060  ; 1037          BX10.D0
U 0724, 7267,5000,5060  ; 1038          BX02.D0,LOADH
                        ; 1039  =0
U 0726, 7267,0377,6060  ; 1040  F16S.Z: LOADL,COND/OUTREG_MT,J/F16S.Z
U 0727, 7167,4377,7020  ; 1041          OUTREG_LOAD,J/F16.S
                        ; 1042  =
                        ; 1043
```

```
                        ; 1044  .TOC "MBC WRITE: MODE 5      - 9-BIT FORMATTED BINARY TO 8-BIT BINARY"
                        ; 1045
                        ; 1046  ; THE MSB OF EACH 9-BIT INPUT BYTE IS IGNORED.
                        ; 1047
U 0725, 7305,0377,7060  ; 1048  B09.B:  COND/SRC_ACK.XIP_0,J/B09B.X             ;GET BYTE 0
                        ; 1049  =00
U 0730, 7547,4377,7060  ; 1050  B09B.X: J/WRT.END                               ;FULL CYCLE OUT, ALL DONE
U 0731, 7305,0377,7040  ; 1051          COND/SRC_ACK.XIP_0,INREG_LOAD,J/B09B.X  ;NO ACK, XIP
U 0732, 7567,4377,7060  ; 1052          J/WRITE.FAIL                            ;ACK, NO XIP
U 0733, 7347,4377,7040  ; 1053          INREG_LOAD                              ;ACK, XIP
                        ; 1054  =
U 0734, 7357,7400,7660  ; 1055          BX01.D0                         ;get 5 bits
U 0735, 7367,4770,7660  ; 1056          BX06.D5                         ;get 3 bits
U 0736, 7407,4000,5060  ; 1057          BX10.D0,LOADH                   ;get 8 bits
                        ; 1058  =0
U 0740, 7407,0377,6060  ; 1059  B09B.Z: LOADL,COND/OUTREG_MT,J/B09B.Z
U 0741, 7257,4377,7020  ; 1060          OUTREG_LOAD,J/B09.B
                        ; 1061  =
                        ; 1062
```

```
                        ; 1063  .TOC "MBC WRITE: END CODE    - ALL WRITE MODES"
                        ; 1064
                        ; 1065  ; THESE ARE THE END ROUTINES FOR ALL OF THE WRITE MODES.
                        ; 1066
                        ; 1067  ;=====  GENERATE AN OUTPUT ONLY IF THE BYTE COUNT IS NOT COMPLETELY ZERO
                        ; 1068  =00
                        ; 1069  PABA.EXIT:
U 0744, 7547,4377,7060  ; 1070          J/WRT.END                               ;3 OR 4 BYTES OUT, EXIT
U 0745, 7477,4377,7060  ; 1071          J/WRT.END.O                             ;5 BYTES OUT (EXACTLY)
U 0746, 7547,4377,7060  ; 1072          J/WRT.END                               ;3 OR 4 BYTES OUT, EXIT
                        ; 1073  WRT.END.O:
U 0747, 7427,6000,7060  ; 1074          BZERO.D0                                ;get 8 bits of zero
                        ; 1075  =
                        ; 1076  ;=====  GENERATE AN OUTPUT FOR THE LAST POSSIBLE WORD
                        ; 1077  =0
U 0742, 7427,0377,6060  ; 1078  WRT.END.Z:      LOADL,COND/OUTREG_MT,J/WRT.END.Z
U 0743, 7377,4377,7020  ; 1079                  OUTREG_LOAD
                        ; 1080  =
U 0737, 0007,4377,7160  ; 1081                  END_XFER,J/IDLE
                        ; 1082
                        ; 1083  ;=====  GENERATE AN OUTPUT WITH 12 LSB'S OF ZERO (1 36-BIT WORD OUT)
U 0750, 7517,6360,7060  ; 1084  WE.1B36:BZERO.D4                ;get 4 bits of zero
U 0751, 7527,6000,5060  ; 1085          BZERO.D0,LOADH          ;load. get 8 bits
                        ; 1086  =0
U 0752, 7527,0377,6060  ; 1087  WE.1.Z: LOADL,COND/OUTREG_MT,J/WE.1.Z           ;WAIT FOR OUTREG EMPTY
U 0753, 7547,4377,7020  ; 1088          OUTREG_LOAD             ;LOAD WORD 2
                        ; 1089  =
U 0754, 0007,4377,7160  ; 1090  WRT.END:        END_XFER,J/IDLE         ;RETURN TO IDLE
                        ; 1091
                        ; 1092  ;=====  GENERATE AN OUTPUT WITH 4 LSB'S OF ZERO (2 36-BIT WORDS OUT)
U 0755, 7527,6360,7060  ; 1093  WE.2B36:BZERO.D4,J/WE.1.Z       ;get 4 bits of zero
                        ; 1094                                  ;then go to common code to output the last word
                        ; 1095
                        ; 1096
                        ; 1097  ;=====  THE COMMON FAIL ROUTINE FOR ALL WRITE MODES
U 0756, 0007,4377,7160  ; 1098  WRITE.FAIL:     END_XFER,J/IDLE ;COMMON FAIL CODE FOR LOGIC ANALYZER TRIGGER
                        ; 1099
                        ; 1100


; NUMBER OF MICRO WORDS USED:
;       D WORDS= 0
;       U WORDS= 495

        END
```

; CROSS REFERENCE LISTING

```
(U) BASH                133 #   306   307   328   329   330   331   338   341   343   344   368
                        369     386   387   388   389   398   401   404   405   423   424   425
                        426     435   438   441   442   451   454   456   457   458   467   468
                        471     473   480   483   485   492   493   494   497   499   506   507
                        510     512   519   520   521   522   525   534   535   552   553   554
                        555     562   563   566   568   569   576   580   582   589   590   591
                        592     595   597   604   605   623   624   642   643   645   718   719
                        731     732   753   754   755   756   761   768   769   791   792   809
                        810     811   812   817   824   825   845   846   847   848   853   861
                        862     867   875   876   877   880   889   890   893   902   903   906
                        908     909   917   920   922   930   933   935   936   944   947   949
                        967     968   969   974   981   982   985   987   994   995   998   1000
                        1007    1010  1017  1019  1037  1038  1055  1056  1057
(U) BYTE_SEL            131 #   262   267   304   306   307   328   329   330   331   338   341
                        343     344   345   366   368   369   386   387   388   389   398   401
                        404     405   406   423   424   425   426   435   438   441   442   451
                        454     456   457   458   467   468   471   473   480   483   485   492
                        493     494   497   499   506   507   510   512   519   520   521   522
                        525     534   535   552   553   554   555   562   563   566   568   569
                        576     577   580   582   589   590   591   592   595   597   604   605
                        606     621   623   624   640   642   643   644   645   661   664   666
                        668     670   672   674   677   679   681   683   687   688   689   718
                        719     731   732   753   754   755   756   761   768   769   791   792
                        809     810   811   812   817   824   825   845   846   847   848   853
                        861     862   867   875   876   877   880   889   890   893   902   903
                        906     908   909   917   920   922   930   933   935   936   944   947
                        949     965   967   968   969   974   981   982   985   987   994   995
                        998     1000  1007  1010  1017  1019  1037  1038  1055  1056  1057  1074
                        1084    1085  1093
(U) COND                71 #
    HDR_MODE            80 #    312   350   737   774
    JUMP_ADRS           83 #
    LAST_ACK            81 #
    MODE                72 #    253   258   314   352   739   776
    OUTREG_MT           82 #    309   340   347   371   400   408   437   453   470   482   496
                        509     524   537   565   579   594   608   626   647   690   692   721
                        734     758   771   794   814   827   850   864   879   892   905   919
                        932     946   951   971   984   997   1009  1021  1040  1059  1078  1087
    SRC_ACK.HDR_M       76 #    254   259
    SRC_ACK.XIP_0       77 #    290   291   292   293   693   714   724   727   740   749   761
                        764     777   784   787   802   805   817   820   830   960   963   974
                        977     989   1000  1003  1012  1030  1033  1048  1051
    SRC_ACK.XIP_1       78 #    284   285   286   287   302   315   324   331   334   353   361
                        364     379   382   389   394   416   419   426   431   442   447   458
                        463     473   476   485   488   499   502   512   515   527   530   545
                        548     555   558   569   572   582   585   597   600   616   619   635
                        638     838   841   853   857   867   871   882   885   895   898   909
                        913     922   926   936   940   987   990
    WRT.XIP.HDR         74 #    251   252   256   257
(U) END_XFER            114 #
    F                   115 #
    T                   116 #   696   705   1081  1090  1098
(U) INREG_LOAD          104 #
    F                   105 #
    T                   106 #   302   304   324   326   334   336   364   366   382   384   394
```

; CROSS REFERENCE LISTING

```
                        396     419     421     431     433     447     449     463     465     476     478     488
                        490     502     504     515     517     530     532     548     550     558     560     572
                        574     585     587     600     602     619     621     638     640     714     716     727
                        729     749     751     764     766     787     789     805     807     820     822     841
                        843     857     859     871     873     885     887     898     900     913     915     926
                        928     940     942     963     965     977     979     990     992     1003    1005    1033
                        1035    1051    1053
(U) J                   67 #
        B.B09           268     635 #   648
        B.B36           264     269     416 #   538
        B.F16           262     361 #   372
        B.F32           263     379 #   409
        B.S16           267     616 #   627
        B09.B           279     1048 #  1060
        B09B.X          1048    1050 #  1051
        B09B.Z          1059 #  1059
        B36.B           275     280     838 #   952
        B36B.X          838     840 #   841
        B36B.Z          850 #   850
        B36B1.X         853     856 #   857
        B36B1.Z         864 #   864
        B36B2.X         867     870 #   871
        B36B2.Z         879 #   879
        B36B3.X         882     884 #   885
        B36B3.Z         892 #   892
        B36B4.X         895     897 #   898
        B36B4.Z         905 #   905
        B36B5.X         909     912 #   913
        B36B5.Z         919 #   919
        B36B6.X         922     925 #   926
        B36B6.Z         932 #   932
        B36B7.X         936     939 #   940
        B36B7.Z         946 #   946
        B36B8.Z         951 #   951
        BA.PA           265     270     545 #   609
        BAPA.X          545     547 #   548
        BAPA1.X         555     557 #   558
        BAPA1.Z         565 #   565
        BAPA2.X         569     571 #   572
        BAPA2.Z         579 #   579
        BAPA3.X         582     584 #   585
        BAPA3.Z         594 #   594
        BAPA4.X         597     599 #   600
        BAPA4.Z         608 #   608
        BB09.X          635     637 #   638
        BB09.Z          647 #   647
        BB36.X          416     418 #   419
        BB361.X         426     430 #   431
        BB361.Z         437 #   437
        BB362.X         442     446 #   447
        BB362.Z         453 #   453
        BB363.X         458     462 #   463
        BB363.Z         470 #   470
        BB364.X         473     475 #   476
        BB364.Z         482 #   482
```

; CROSS REFERENCE LISTING

```
BB365.X     485     487 #   488
BB365.Z     496 #   496
BB366.X     499     501 #   502
BB366.Z     509 #   509
BB367.X     512     514 #   515
BB367.Z     524 #   524
BB368.X     527     529 #   530
BB369.Z     537 #   537
BF16.X      361     363 #   364
BF16.Z      371 #   371
BF32.X      379     381 #   382
BF321.X     389     393 #   394
BF321.Z     400 #   400
BF322.Z     408 #   408
BS16.X      616     618 #   619
BS16.Z      626 #   626
F16.B       273     784 #   795
F16.S       278     1030 #  1041
F16B.X      784     786 #   787
F16B.Z      794 #   794
F16S.X      1030    1032 #  1033
F16S.Z      1040 #  1040
F32.B       274     802 #
F32B.X      802     804 #   805     830
F32B.Z      814 #   814
F32B1.X     817     819 #   820
F32B1.Z     827 #   827
HR.SEL      254     284 #
HR16.HDR.X  312     314 #
HR16.X      284     286     301 #   302     315
HR16.Z      309 #   309
HR32.HDR.X  350     352 #
HR32.X      285     287     323 #   324     353
HR321.X     331     333 #   334
HR321.Z     340 #   340
HR322.Z     347 #   347
HW.SEL      259     290 #
HW16.HDR.X  737     739 #
HW16.X      290     292     713 #   714     740
HW16.Z      721 #   721
HW161.X     724     726 #   727
HW161.Z     734 #   734
HW32.HDR.X  774     776 #
HW32.X      291     293     748 #   749     777
HW32.Z      758 #   758
HW321.X     761     763 #   764
HW321.Z     771 #   771
IDLE        251 #   251     252     256     257     696     705     1081    1090    1098
MR.X        253     262 #   314     352
MW.X        258     273 #   739     776
PA.BA       276     281     960 #   1022
PABA.EXIT   989     1069 #
PABA.X      960     962 #   963
PABA.Z      971 #   971
PABA1.X     974     976 #   977
```

; CROSS REFERENCE LISTING

```
        PABA1.Z                 984 #   984
        PABA2.X                 987     989 #   990
        PABA2.Z                 997 #   997
        PABA3.X                 1000    1002 #  1003
        PABA3.Z                 1009 #  1009
        PABA4.X                 1014 #
        PABA4.Y                 1015    1017 #
        PABA4.Z                 1021 #  1021
        READ.FAIL               303     325     335     365     383     395     420     432     448     464     477     489
                                503     516     531     549     559     573     586     601     620     639     699     705 #
        RX                      687 #   697     700
        RX.B.X                  693     696 #
        RX.H                    664     666     668     670     672     674     689 #
        RX.L                    677     679     681     683     690 #
        RX.L.X                  690     692 #   692
        RX.X                    661     688 #
        RX0H                    529     664 #
        RX0L                    475     571     677 #
        RX1H                    599     666 #
        RX2H                    514     668 #
        RX2L                    462     679 #
        RX4H                    501     670 #
        RX4L                    446     557     681 #
        RX5H                    584     672 #
        RX6H                    487     674 #
        RX6L                    333     393     430     683 #
        RX6X                    301     323     363     381     418     547     618     637     661 #
        WE.1.Z                  1087 #  1087    1093
        WE.1B36                 870     1084 #
        WE.2B36                 925     1093 #
        WRITE.FAIL              715     728     750     765     788     806     821     842     858     872     886     899
                                914     927     941     964     978     991     1004    1034    1052    1098 #
        WRT.END                 713     726     748     763     786     804     819     840     856     884     962     976
                                1002    1014    1016    1032    1050    1070    1072    1090 #
        WRT.END.O               897     912     939     1071    1073 #
        WRT.END.Z               1078 #  1078
(U) LOADH                       123 #   307     331     344     369     389     405     426     442     458     475     480
                                493     507     521     535     555     571     576     590     605     624     645     689
                                719     732     755     769     792     811     825     847     862     870     876     890
                                895     909     922     936     949     969     982     989     994     1000    1017    1038
                                1057    1085
(U) LOADL                       120 #   309     340     347     371     400     408     437     453     470     482     496
                                509     524     537     565     579     594     608     626     647     690     721     734
                                758     771     794     814     827     850     864     879     892     905     919     925
                                932     946     951     971     984     997     1009    1021    1040    1059    1078    1087
(U) LOADX                       126 #   306     329     343     368     387     404     424     441     456     473     485
                                499     512     527     553     568     582     597     623     643     688
(U) MACRO%
        B00.D0                  198 #   597
        B01.D0                  199 #   458     512
        B01.D6                  200 #   525
        B02.D6                  201 #   595
        B03.D0                  202 #   344     405     442     499     555
        B03.D6                  203 #   457     510     522
        B03.D7                  204 #   592
```

; CROSS REFERENCE LISTING

```
        B04.D0                  205 #   582
        B05.D0                  206 #   331     389     426     485
        B05.D6                  207 #   497
        B06.D0                  208 #   605
        B06.D1                  209 #   569
        B06.D4                  210 #   591
        B06.D5                  211 #   554
        B06.D6                  212 #   580
        B07.D0                  213 #   307     369     473     535     623     645
        B07.D6                  214 #   330     388     425     483     494
        B08.D0                  215 #   568
        B09.D0                  216 #   456     521
        B09.D6                  217 #   471
        B10.D6                  218 #   566
        B11.D0                  219 #   343     404     441     507     590
        B11.D6                  220 #   454     468     520
        B11.D7                  221 #   563
        B12.D0                  222 #   553
        B13.D0                  223 #   329     387     424     493
        B13.D6                  224 #   341     401     438
        B14.D0                  225 #   576     643
        B14.D1                  226 #   604
        B14.D4                  227 #   562
        B14.D5                  228 #   589
        B14.D6                  229 #   552
        B15.D0                  230 #   306     368     480     534     624
        B15.D2                  231 #   467     519
        B15.D4                  232 #   451     506
        B15.D6                  233 #   328     338     386     398     423     435     492
        B15.D7                  234 #   642
        BX00.D0                 163 #   753     809     845     902
        BX00.D1                 164 #   967     994
        BX00.D4                 165 #   875     930
        BX01.D0                 166 #   1055
        BX02.D0                 167 #   718     731     791     1038
        BX04.D0                 168 #   876     933
        BX06.D5                 169 #   1056
        BX06.D6                 170 #   754     810     846     903
        BX06.D7                 171 #   968     995
        BX07.D1                 172 #   969     998
        BX08.D0                 173 #   755     811     847     906
        BX08.D4                 174 #
        BX10.D0                 175 #   719     732     792     1037    1057
        BX10.D6                 176 #   877     935
        BX12.D0                 177 #   880     936
        BX14.D1                 178 #   974     1000
        BX14.D2                 179 #
        BX14.D6                 180 #   756     812     848     908
        BX16.D0                 181 #   761     817     853     909
        BX18.D2                 182 #   768     824     861     917
        BX18.D5                 183 #   981     1007
        BX18.D6                 184 #   889     944
        BX20.D0                 185 #   890     947
        BX21.D1                 186 #   982     1010
        BX24.D0                 187 #   769     825     862     920
```

; CROSS REFERENCE LISTING

```
BX28.D0          188 #  893   949
BX28.D1          189 #  985   1017
BX32.D0          190 #  867   922
BX32.D5          191 #  987   1019
BZERO.D0         239 #  664   677   688   689   965   1074  1085
BZERO.D1         240 #  666
BZERO.D2         241 #  668   679
BZERO.D4         242 #  345   406   670   681   1084  1093
BZERO.D5         243 #  672
BZERO.D6         244 #  262   267   304   366   621   640   661   674   683   687
BZERO.D7         245 #  577   606   644
END_XFER         117 #  696   705   1081  1090  1098
INREG_LOAD       107 #  302   304   324   326   334   336   364   366   382   384   394
                 396    419   421   431   433   447   449   463   465   476   478   488
                 490    502   504   515   517   530   532   548   550   558   560   572
                 574    585   587   600   602   619   621   638   640   714   716   727
                 729    749   751   764   766   787   789   805   807   820   822   841
                 843    857   859   871   873   885   887   898   900   913   915   926
                 928    940   942   963   965   977   979   990   992   1003  1005  1033
                 1035   1051  1053
LOADH            124 #  307   331   344   369   389   405   426   442   458   475   480
                 493    507   521   535   555   571   576   590   605   624   645   689
                 719    732   755   769   792   811   825   847   862   870   876   890
                 895    909   922   936   949   969   982   989   994   1000  1017  1038
                 1057   1085
LOADL            121 #  309   340   347   371   400   408   437   453   470   482   496
                 509    524   537   565   579   594   608   626   647   690   721   734
                 758    771   794   814   827   850   864   879   892   905   919   925
                 932    946   951   971   984   997   1009  1021  1040  1059  1078  1087
LOADX            127 #  306   329   343   368   387   404   424   441   456   473   485
                 499    512   527   553   568   582   597   623   643   688
OUTREG_LOAD      112 #  310   341   348   372   401   409   438   454   471   483   497
                 510    525   538   566   580   595   609   627   648   697   700   722
                 735    759   772   795   815   828   851   865   880   893   906   920
                 933    947   952   972   985   998   1010  1022  1041  1060  1079  1088
SAVE0            138 #  306   307   329   331   343   344   368   369   387   389   404
                 405    424   426   441   442   456   458   473   480   485   493   499
                 507    512   521   534   535   553   555   568   576   582   590   597
                 605    623   624   643   645   664   677   688   689   718   719   731
                 732    753   755   761   769   791   792   809   811   817   825   845
                 847    853   862   867   876   880   890   893   902   906   909   920
                 922    933   936   947   949   965   1037  1038  1055  1057  1074  1085
SAVE1            139 #  569   604   666   967   969   974   982   985   994   998   1000
                 1010   1017
SAVE2            140 #  467   519   668   679   768   824   861   917
SAVE3            141 #
SAVE4            142 #  345   406   451   506   562   591   670   681   875   930   1084
                 1093
SAVE5            143 #  554   589   672   981   987   1007  1019  1056
SAVE6            144 #  262   267   304   328   330   338   341   366   386   388   398
                 401    423   425   435   438   454   457   468   471   483   492   494
                 497    510   520   522   525   552   566   580   595   621   640   661
                 674    683   687   754   756   810   812   846   848   877   889   903
                 908    935   944
SAVE7            145 #  563   577   592   606   642   644   968   995
```

; CROSS REFERENCE LISTING

```
(U) OUTREG_LOAD          109 #
        F                110 #
        T                111 #   310   341   348   372   401   409   438   454   471   483   497
                         510     525   538   566   580   595   609   627   648   697   700   722
                         735     759   772   795   815   828   851   865   880   893   906   920
                         933     947   952   972   985   998  1010  1022  1041  1060  1079  1088
(U) SAVE                 137 #   262   267   304   306   307   328   329   330   331   338   341
                         343     344   345   366   368   369   386   387   388   389   398   401
                         404     405   406   423   424   425   426   435   438   441   442   451
                         454     456   457   458   467   468   471   473   480   483   485   492
                         493     494   497   499   506   507   510   512   519   520   521   522
                         525     534   535   552   553   554   555   562   563   566   568   569
                         576     577   580   582   589   590   591   592   595   597   604   605
                         606     621   623   624   640   642   643   644   645   661   664   666
                         668     670   672   674   677   679   681   683   687   688   689   718
                         719     731   732   753   754   755   756   761   768   769   791   792
                         809     810   811   812   817   824   825   845   846   847   848   853
                         861     862   867   875   876   877   880   889   890   893   902   903
                         906     908   909   917   920   922   930   933   935   936   944   947
                         949     965   967   968   969   974   981   982   985   987   994   995
                         998    1000  1007  1010  1017  1019  1037  1038  1055  1056  1057  1074
                        1084    1085  1093
```

; LOCATION / LINE NUMBER INDEX
; DCODE LOC'N 0 1 2 3 4 5 6 7

d 0000

; LOCATION / LINE NUMBER INDEX

; UCODE LOC'N	0	1	2	3	4	5	6	7
u 0000	251	252	253	254	256	257	258	259
u 0010	262	263	264	265	267	268	269	270
u 0020	273	274	275	276	278	279	280	281
u 0030	284	285	286	287	290	291	292	293
u 0040	301	302	303	304	306	307	309	310
u 0050	312	328	314	315	323	324	325	326
u 0060	329	330	331	338	333	334	335	336
u 0070	340	341	343	344	345	350	347	348
u 0100	352	353	361	368	363	364	365	366
u 0110	369	379	371	372	381	382	383	384
u 0120	386	387	388	389	393	394	395	396
u 0130	398	404	400	401	405	406	408	409
u 0140	416	423	424	425	418	419	420	421
u 0150	426	435	437	438	430	431	432	433
u 0160	441	442	451	456	446	447	448	449
u 0170	453	454	457	458	462	463	464	465
u 0200	467	468	470	471	473	480	482	483
u 0210	475	476	477	478	485	492	493	494
u 0220	487	488	489	490	496	497	499	506
u 0230	501	502	503	504	507	512	509	510
u 0240	514	515	516	517	519	520	521	522
u 0250	524	525	527	534	529	530	531	532
u 0260	535	545	537	538	547	548	549	550
u 0270	552	553	554	555	557	558	559	560
u 0300	562	563	565	566	568	569	576	577
u 0310	571	572	573	574	579	580	582	589
u 0320	584	585	586	587	590	591	592	597
u 0330	594	595	604	605	599	600	601	602
u 0340	606	616	608	609	618	619	620	621
u 0350	623	624	626	627	635	642	643	644
u 0360	637	638	639	640	645	661	647	648
u 0370	664	666	668	670	672	674	677	679
u 0400	681	683	687	688	689	690	692	693
u 0410	696	697	699	700	705	718	719	724
u 0420	713	714	715	716	721	722	731	732
u 0430	726	727	728	729	734	735	737	753
u 0440	739	740	754	755	748	749	750	751
u 0450	756	761	758	759	763	764	765	766
u 0460	768	769	771	772	774	784	776	777
u 0470	786	787	788	789	791	792	794	795
u 0500	802	809	810	811	804	805	806	807
u 0510	812	817	814	815	819	820	821	822
u 0520	824	825	827	828	830	838	845	846
u 0530	840	841	842	843	847	848	850	851
u 0540	853	861	862	867	856	857	858	859
u 0550	864	865	875	876	870	871	872	873
u 0560	877	882	879	880	884	885	886	887
u 0570	889	890	892	893	895	902	903	908

; LOCATION / LINE NUMBER INDEX

; UCODE LOC'N	0	1	2	3	4	5	6	7
u 0600	897	898	899	900	905	906	909	917
u 0610	912	913	914	915	919	920	922	930
u 0620	925	926	927	928	932	933	935	936
u 0630	939	940	941	942	944	949	946	947
u 0640	951	952	960	967	962	963	964	965
u 0650	968	969	971	972	974	981	982	987
u 0660	976	977	978	979	984	985	994	995
u 0670	989	990	991	992	997	998	1000	1007
u 0700	1002	1003	1004	1005	1009	1010	1012	1019
u 0710	1014	1015	1016	1017	1021	1022	1030	1037
u 0720	1032	1033	1034	1035	1038	1048	1040	1041
u 0730	1050	1051	1052	1053	1055	1056	1057	1081
u 0740	1059	1060	1078	1079	1070	1071	1072	1074
u 0750	1084	1085	1087	1088	1090	1093	1098	

no errors detected
END OF MICRO CODE ASSEMBLY
used 4.69 seconds

; TABLE OF CONTENTS

```
; 1     ;<MEIS.VEX>VEX02.MIC.2, 20-Feb-85 22:32:03, Edit by LOUGHEED
; 2     ; In End Massbus Read (R12) do a CLEAR_XIPR in case we received
; 3     ;  a shutdown request before RX to Mport (V16) completed.
; 4     ;<MEIS.VEX>VEX01.MIC.94, 17-Jul-84 17:05:56, Edit by LOUGHEED
; 5     ;Move first instruction of the IDLE routine into the vector dispatch table
; 6     ;RX packet flush exits early if there is no packet to flush
; 7     ;<MEIS.VEX>VEX01.MIC.93, 17-Jul-84 16:28:36, Edit by LOUGHEED
; 8     ;End Massbus Read is now the same as RX packet flush.  This causes Massbus
; 9     ; reads that do not read the entire packet to leave the pointers in a good
; 10    ; state.  Eliminates spurious runt and giant packets.
; 11    ;<MEIS.VEX>VEX01.MIC.92, 17-Jul-84 15:47:12, Edit by LOUGHEED
; 12    ;Move a SET_STATUS,RX_PKT_AVAIL from V13WRD to V13WRE so we don't get a
; 13    ; spurious packet available in case we decide to abort at V13WRE+1.
; 14    ;<MEIS.VEX>VEX01.MIC.91, 12-May-84 18:17:22, Edit by LOUGHEED
; 15    ;First production version of Via Board ucode - VEX01
; 16
; 17
```

```
; 18    .TITLE "MEIS 10MB ETHERNET DATA UCODE - GFS/KSL"
; 19
; 20    ;"This time for sure!"
; 21
; 22    ;Write PROM files using WVEX1.SAI.
; 23
; 24    ;  MONUMENT: Data alignment within 36-bit words.
; 25    ;
; 26    ;       There are 14 bytes of header encapsulation for the 10MB Ethernet.
; 27    ; This comes to 3.5 36-bit words.  To force an extra half-word of padding
; 28    ; so that a datagram's body starts on a 36-bit word boundary, we must do the
; 29    ; following things.
; 30    ;       On a write we assume that the user has supplied the half-word of
; 31    ; padding.  We read it into our buffer, but when it comes to sending data to
; 32    ; the NI, we do NOT send that first 16-bit word.  This amounts to incrementing
; 33    ; TWCUR an extra time in the IDLE loop startup code.
; 34    ;       On a read from the NI, we take everything.  When the user starts
; 35    ; reading (Massbus or sniffing) we insert an extra 16 bits.  This is done by
; 36    ; an extra MPORT_LOAD in the Start Massbus Read code.  The Packet Status Read
; 37    ; code addes an extra 2 bytes to the count it returns (diagnostic writers
; 38    ; beware of that fact!).
; 39
; 40
```

```
; 41    .TOC "Field Definitions"
; 42
; 43    ;The Jump field is 9 bits (512 words)
; 44    J/=0,9,8,+
; 45
; 46    ;Condition code selection (actual values not yet defined)
; 47    ;  Conditions are active high (H) unless otherwise marked by a (L).
; 48    ;  Dotted symbols are for two bit selection conditions.
; 49
; 50    COND/=10,5,13,D
; 51            ZERO            =00     ;Last ALU result was zero
; 52            NEG             =01     ;Last ALU result was negative
; 53            RX_ENABLE_OUT.IN =02    ;Request to toggle NI board reception
; 54            DBUS_LSB        =03     ;State of LSB of data bus
; 55            RUNNING.ATN_L   =04     ;Product of TX_REQ_ENABLE (H), NI_INTERRUPT (L)
; 56            ZERO.FLUSH      =05     ;Product of ZERO (H) and discard pkt logic (H)
; 57
; 58            TX_AVAIL_L.MBUS_VALID =06 ;TX buffer available (L) and low byte is
; 59                                      ;is valid from mport read (H).  A crock
; 60                                      ;because we ran out of LSB's for the COND.
; 61                                      ;The two signals are unrelated.
; 62
; 63            RX_EOF.LBYTE    =07     ;Product of RX_EOF (L) and Low Byte Valid (L)
; 64
; 65            JUMP_ADDRS      =10     ;Default field select
; 66            RCON_REQ        =20     ;Branch on RCON request
; 67            VECTOR_REQ      =30     ;Branch on vector request
; 68
; 69    JUMP_VECTOR     "J/VECTOR,COND/VECTOR_REQ"
; 70
```

```
; 71
; 72    ;Register allocation in 2901 ALU, A input
; 73
; 74    ALUA/=17,4,31,D
; 75            TWBASE  =00             ;transmit withdrawl edge
; 76            TDBASE  =01             ;transmit deposit base
; 77            TWCUR   =02             ;transmit withdrawl word
; 78            TDCUR   =03             ;transmit deposit word
; 79            TWTOP   =04             ;transmit withdrawl top
; 80            TWRAP   =05             ;transmit wrap address
; 81            TBLEFT  =06             ;available bytes left in transmit buffer
; 82
; 83            RWBASE  =07             ;receiver withdrawl edge
; 84            RDBASE  =10             ;receiver deposit base
; 85            RWCUR   =11             ;receiver withdrawl word
; 86            RDCUR   =12             ;receiver deposit word
; 87            RWTOP   =13             ;receiver withdrawl top
; 88             SNIFF  =13             ;For "sniffing" data to and from the buffer
; 89             ADDRM  =13             ;For setting our Ethernet address
; 90            RWRAP   =14             ;receiver wrap address
; 91            RDLIMIT =15             ;receiver deposit limit
; 92
; 93            MSIZE   =16             ;bytes in a maximum sized packet
; 94            TEMP    =17             ;temporary register
; 95
```

```
; 96
; 97    ;Register allocation in 2901 ALU, B input
; 98
; 99    ALUB/=17,4,27,D
; 100
; 101           TWBASE  =00             ;transmit withdrawl edge
; 102           TDBASE  =01             ;transmit deposit base
; 103           TWCUR   =02             ;transmit withdrawl word
; 104           TDCUR   =03             ;transmit deposit word
; 105           TWTOP   =04             ;transmit withdrawl top
; 106           TWRAP   =05             ;transmit wrap address
; 107           TBLEFT  =06             ;available bytes left in transmit buffer
; 108
; 109           RWBASE  =07             ;receiver withdrawl edge
; 110           RDBASE  =10             ;receiver deposit base
; 111           RWCUR   =11             ;receiver withdrawl word
; 112           RDCUR   =12             ;receiver deposit word
; 113           RWTOP   =13             ;receiver withdrawl top
; 114            SNIFF  =13             ;For "sniffing" data to and from the buffer
; 115            ADDRM  =13             ;For setting our Ethernet address
; 116           RWRAP   =14             ;receiver wrap address
; 117           RDLIMIT =15             ;receiver deposit limit
; 118
; 119           MSIZE   =16             ;bytes in a maximum sized packet
; 120           TEMP    =17             ;temporary register
; 121
```

```
; 122
; 123   ; 2901 function field.  See 2901 documentation for the rest of the opcodes.
; 124
; 125   ALUFN/=47,6,23,D
; 126                                       ;CRY_0          CRY_1
; 127           addAQ   =00                 ; A+Q           A+Q+1
; 128           addAB   =01                 ; A+B           A+B+1
; 129           addZQ   =02                 ; Q             Q+1
; 130           addZB   =03                 ; B             B+1
; 131           addZA   =04                 ; A             A+1
; 132           addDA   =05                 ; D+A           D+A+1
; 133
; 134           subQA   =10                 ; Q-A-1         Q-A
; 135           subBA   =11                 ; B-A-1         B-A
; 136           subZB   =13                 ; B-1           B
; 137           subZA   =14                 ; A-1           A
; 138           subAQ   =20                 ; A-Q-1         A-Q
; 139           subAB   =21                 ; A-B-1         A-B
; 140           subZQ   =22                 ; -Q-1          -Q
; 141           subAZ   =24                 ; -A-1          -A
; 142           subDA   =25                 ; D-A-1         D-A
; 143
; 144           Q       =32                 ; Q
; 145           B       =33                 ; B
; 146           A       =34                 ; A
; 147           D       =37                 ; D
; 148           Z       =47                 ; Z
; 149
; 150           NAandQ  =50                 ;(NOT A) AND Q
; 151           NAandB  =51                 ;(NOT A) AND B
; 152
; 153   ;2901 destination field
; 154
; 155   ALUDEST/=1,3,17,D
; 156           Q       =0                  ;Shift register
; 157           NOP     =1                  ;Do nothing
; 158           BRAMA   =2                  ;Register addressed by B gets values, A to bus
; 159           BRAM    =3                  ;Register addressed by B and data bus get value
; 160           BQDOWN  =4                  ;Right shifted result to B and Q
; 161           BDOWN   =5                  ;Right shifted result to B
; 162           BQUP    =6                  ;Left shifted result to B and Q
; 163           BUP     =7                  ;Left shifted result to B
; 164
; 165   ;2901 ALU control
; 166
; 167   CARRY/=0,1,14,D                     ;Handling of carry bit
; 168           F=1
; 169           T=0
; 170   CRY_0             "CARRY/T"
; 171   CRY_1             "CARRY/F"
; 172
; 173
```

```
; 174
; 175   ;Status control lines.  All are active high except for TX_BUFR_AVAIL_L.
; 176
; 177   TX_ACQ_ERR/=1,1,48,D            ;Error while acquiring transmit buffer
; 178           F=1
; 179           T=0
; 180   TX_ACQ_ERR               "TX_ACQ_ERR/T"
; 181
; 182   RX_OVERFLOW/=1,1,49,D           ;Receiver overflow
; 183           F=1
; 184           T=0
; 185   RX_OVERFLOW              "RX_OVERFLOW/T"
; 186
; 187   TX_BUFR_AVAIL_L/=1,1,50,D              ;Transmit buffer available
; 188           F=1
; 189           T=0
; 190   TX_BUFR_AVAIL_L          "TX_BUFR_AVAIL_L/T"
; 191
; 192   RX_PKT_AVAIL/=1,1,51,D          ;Packet available in receiver buffer
; 193           F=1
; 194           T=0
; 195   RX_PKT_AVAIL             "RX_PKT_AVAIL/T"
; 196
; 197   LBYTE_VALID/=1,1,52,D           ;Low byte of last word is valid (set for NI)
; 198           F=1
; 199           T=0
; 200   LBYTE_VALID              "LBYTE_VALID/T"
; 201
; 202   TX_EOF/=1,1,53,D                ;End of frame being transmitted
; 203           F=1
; 204           T=0
; 205   TX_EOF                    "TX_EOF/T"
; 206
; 207   TX_REQ_ENABLE/=1,1,54,D         ;NI is asking for data
; 208           F=1
; 209           T=0
; 210   TX_REQ_ENABLE             "TX_REQ_ENABLE/T"
; 211
; 212   RX_ENABLE/=1,1,55,D             ;Reception enable
; 213           F=1
; 214           T=0
; 215   RX_ENABLE                 "RX_ENABLE/T"
; 216
; 217   STATUS/=0,1,56,D                ;Set/clear specified status bits
; 218           F=0
; 219           T=1
; 220   CLEAR_STATUS              "STATUS/F"
; 221   SET_STATUS                "STATUS/T"
; 222
```

```
; 223
; 224   ;Data bus read/load control
; 225
; 226   DALU_READ/=1,1,32,D             ;Load results of ALU computation onto data bus
; 227           F=1
; 228           T=0
; 229   DALU_READ                "DALU_READ/T"
; 230   DEBUG                    "DALU_READ/T"
; 231
; 232   SLIDE_READ/=1,1,33,D            ;Read slave ID register
; 233           F=1
; 234           T=0
; 235   SLIDE_READ               "SLIDE_READ/T"
; 236
; 237   BUFR_READ/=1,1,34,D             ;Read RAM addressed by Dbus
; 238           F=1
; 239           T=0
; 240   BUFR_READ                "BUFR_READ/T"
; 241
; 242   BUFR_LOAD/=1,1,35,D             ;Write RAM addressed by Dbus
; 243           F=1
; 244           T=0
; 245   BUFR_LOAD                "BUFR_LOAD/T"
; 246
; 247   XPORT_READ/=1,1,36,D            ;Read port to register board
; 248           F=1
; 249           T=0
; 250   XPORT_READ               "XPORT_READ/T"
; 251
; 252   XPORT_LOAD/=1,1,37,D            ;Write port to register board
; 253           F=1
; 254           T=0
; 255   XPORT_LOAD               "XPORT_LOAD/T"
; 256
; 257   MPORT_READ/=1,1,38,D            ;Read port to barrel board
; 258           F=1
; 259           T=0
; 260   MPORT_READ               "MPORT_READ/T"
; 261
; 262   MPORT_LOAD/=1,1,39,D            ;Write port to barrel board
; 263           F=1
; 264           T=0
; 265   MPORT_LOAD               "MPORT_LOAD/T"
; 266
```

```
; 267
; 268   ;NI read/load control fields
; 269
; 270   NI_BADRS_RST/=1,1,40,D          ;Reset Bad Address flag in NI
; 271           F=1
; 272           T=0
; 273   NI_BADRS_RST             "NI_BADRS_RST/T"
; 274
; 275   NI_MASK_ADRS_LOAD/=1,1,41,D     ;Load address for NI Mask read or write
; 276           F=1
; 277           T=0
; 278   NI_MASK_ADRS_LOAD        "NI_MASK_ADRS_LOAD/T"
; 279
; 280   NI_MASK_DATA_READ/=1,1,42,D     ;Read data in address mask of NI
; 281           F=1
; 282           T=0
; 283   NI_MASK_DATA_READ        "NI_MASK_DATA_READ/T"
; 284
; 285   NI_MASK_DATA_LOAD/=1,1,43,D     ;Write data in address mask of NI
; 286           F=1
; 287           T=0
; 288   NI_MASK_DATA_LOAD        "NI_MASK_DATA_LOAD/T"
; 289
; 290   NI_DATA_READ/=1,1,44,D          ;Read word of data from NI
; 291           F=1
; 292           T=0
; 293   NI_DATA_READ             "NI_DATA_READ/T"
; 294
; 295   NI_DATA_LOAD/=1,1,45,D          ;Write word of data to NI
; 296           F=1
; 297           T=0
; 298   NI_DATA_LOAD             "NI_DATA_LOAD/T"
; 299
; 300   NI_COMMAND_READ/=1,1,46,D       ;Read status from NI
; 301           F=1
; 302           T=0
; 303   NI_COMMAND_READ          "NI_command_read/T"
; 304
; 305   NI_COMMAND_LOAD/=1,1,47,D       ;Send command to NI
; 306           F=1
; 307           T=0
; 308   NI_COMMAND_LOAD          "NI_COMMAND_LOAD/T"
; 309
; 310   CLEAR_XIPR/=0,1,60,D            ;Clear transfer in progress
; 311           F=0
; 312           T=1
; 313   CLEAR_XIPR               "CLEAR_XIPR/T"
; 314
; 315
```

```
; 316
; 317   ;NI register read/write control
; 318
; 319   NI_REG_SEL/=7,3,63,D
; 320
; 321           RX_REG  =06
; 322           TX_REG  =07
; 323
; 324   NI_RX_STATUS_READ       "NI_REG_SEL/RX_REG,NI_COMMAND_READ"
; 325
; 326   NI_TX_STATUS_READ       "NI_REG_SEL/TX_REG,NI_COMMAND_READ"
; 327
; 328   NI_RX_COMMAND_LOAD      "NI_REG_SEL/RX_REG,NI_COMMAND_LOAD"
; 329
; 330   NI_TX_COMMAND_LOAD      "NI_REG_SEL/TX_REG,NI_COMMAND_LOAD"
; 331
; 332   ;Magic numbers for PROM patching.
; 333   ;
; 334   ;     Bit Range      VEX prom
; 335   ;       00-07           0
; 336   ;       08-15           1
; 337   ;       16-23           2
; 338   ;       24-31           3
; 339   ;       32-39           4
; 340   ;       40-47           5
; 341   ;       48-55           6
; 342   ;       56-63           7
; 343
; 344
```

```
                                    ; 345   .TOC "Initialization and Idle Loop"
                                    ; 346
                                    ; 347   ;INIT - Set initial values
                                    ; 348   ;-------------------------
                                    ; 349
                                    ; 350   ;First initialize the Registers.  We need to have:
                                    ; 351   ;
                                    ; 352   ;       TWRAP   10000                   ;Wrap point of TX buffer
                                    ; 353   ;       TWBASE  0                       ;TX withdraw base
                                    ; 354   ;       TDBASE  0                       ;TX deposit base
                                    ; 355   ;       TDCUR   1                       ;TX deposit current
                                    ; 356   ;       TWCUR   0                       ;TX withdraw current
                                    ; 357   ;       MSIZE   2000                    ;Max. size of a packet (2048.)
                                    ; 358   ;       TBLEFT  TWRAP-MSIZE-1           ;Bytes left in TX buffer
                                    ; 359   ;
                                    ; 360   ;       RWRAP   20000                   ;Wrap point of RX buffer
                                    ; 361   ;       RWBASE  TWRAP                   ;RX withdraw base
                                    ; 362   ;       RDBASE  TWRAP                   ;RX deposit base
                                    ; 363   ;       RDCUR   TWRAP+1                 ;RX deposit current
                                    ; 364   ;       RWCUR   TWRAP+1                 ;RX withdraw current
                                    ; 365   ;       RDLIMIT RWRAP-1                 ;RX deposit limit
                                    ; 366   ;
                                    ; 367   ;The registers RWTOP, TEMP, and TWTOP do not need initial values.
                                    ; 368   ;Note well:  TWRAP = length of RX buffer.  This assumption is used in V13.
                                    ; 369
                                    ; 370   INIT:   NI_BADRS_RST,CLEAR_STATUS,TX_REQ_ENABLE,TX_EOF,RX_PKT_AVAIL,
U 0000, 0012,0147,7777,7577,0000,3400 ; 371            TX_BUFR_AVAIL_L,RX_OVERFLOW,TX_ACQ_ERR,LBYTE_VALID,RX_ENABLE
                                    ; 372
                                    ; 373   ;Now do the easy registers
                                    ; 374
                                    ; 375   ;TWBASE := 0
U 0001, 0022,0347,0367,7777,7760,3400 ; 376           DEBUG,ALUA/TEMP,ALUB/TWBASE,ALUFN/Z,ALUDEST/BRAM
                                    ; 377   ;TDBASE := 0
U 0002, 0032,0347,0767,7777,7760,3400 ; 378           DEBUG,ALUA/TEMP,ALUB/TDBASE,ALUFN/Z,ALUDEST/BRAM
                                    ; 379   ;TWCUR := 0
U 0003, 0042,0347,1367,7777,7760,3400 ; 380           DEBUG,ALUA/TEMP,ALUB/TWCUR,ALUFN/Z,ALUDEST/BRAM
                                    ; 381   ;TDCUR := 1
U 0004, 0052,1304,1447,7777,7760,3400 ; 382           DEBUG,ALUA/TWCUR,ALUB/TDCUR,ALUFN/ADDZA,ALUDEST/BRAM,CRY_1
                                    ; 383
                                    ; 384   ;Begin setting up the wrap registers. We do RWRAP, TWRAP, then MSIZE.
                                    ; 385
                                    ; 386   ;Start by rotating a 1 into the MSB of RWRAP to get 100,000
U 0005, 0062,0534,6067,7777,7760,3400 ; 387           DEBUG,ALUA/TDCUR,ALUB/RWRAP,ALUFN/A,ALUDEST/BDOWN
                                    ; 388   ;Shift down to get 40,000.
U 0006, 0072,0534,6307,7777,7760,3400 ; 389           DEBUG,ALUA/RWRAP,ALUB/RWRAP,ALUFN/A,ALUDEST/BDOWN
                                    ; 390   ;Shift down to get 20,000, the final value of RWRAP
U 0007, 0102,0534,6307,7777,7760,3400 ; 391           DEBUG,ALUA/RWRAP,ALUB/RWRAP,ALUFN/A,ALUDEST/BDOWN
                                    ; 392   ;Shift down to get 10,000, the final value of TWRAP
U 0010, 0112,0534,2707,7777,7760,3400 ; 393           DEBUG,ALUA/RWRAP,ALUB/TWRAP,ALUFN/A,ALUDEST/BDOWN
                                    ; 394   ;Shift down to get 4,000
U 0011, 0122,0534,7127,7777,7760,3400 ; 395           DEBUG,ALUA/TWRAP,ALUB/MSIZE,ALUFN/A,ALUDEST/BDOWN
                                    ; 396   ;Shift down to get 2,000, the final value of MSIZE (= 2048. bytes)
U 0012, 0132,0534,7347,7777,7760,3400 ; 397           DEBUG,ALUA/MSIZE,ALUB/MSIZE,ALUFN/A,ALUDEST/BDOWN
                                    ; 398
```

```
                                    ; 399
                                    ; 400   ;We interested in receiving interrupts on TX underflow and TX timeout.
                                    ; 401   ; Must write a 5 to the NI TX command register. Shift a 1 twice to get a 4.
U 0013, 0142,0734,7467,7777,7760,3400   ; 402           DEBUG,ALUA/TDCUR,ALUB/TEMP,ALUFN/A,ALUDEST/BUP
U 0014, 0152,0734,7767,7777,7760,3400   ; 403           DEBUG,ALUA/TEMP,ALUB/TEMP,ALUFN/A,ALUDEST/BUP
                                    ; 404
                                    ; 405   ;Add 1 and 4 to get 5, write to NI TX command register
                                    ; 406           DALU_READ,NI_TX_COMMAND_LOAD,ALUA/TEMP,ALUB/TDCUR,ALUFN/ADDAB,
U 0015, 0162,0101,1767,7776,7760,3400   ; 407            ALUDEST/NOP
                                    ; 408
                                    ; 409   ;Disable the NI receiver by writing a zero to the NI RX command register
U 0016, 0172,0147,7767,7776,7760,3000   ; 410           DALU_READ,NI_RX_COMMAND_LOAD,ALUA/TEMP,ALUB/TEMP,ALUFN/Z,ALUDEST/NOP
                                    ; 411
                                    ; 412   ;Do an NI read to make sure we don't get a garbage word after a MEIS reset.
                                    ; 413   ; For good measure we also make sure that TX_REQ_ENABLE is down and that
                                    ; 414   ; the bad address line is also reset.
U 0017, 0202,0147,7777,7567,7720,3400   ; 415           CLEAR_STATUS,TX_REQ_ENABLE,NI_DATA_READ,NI_BADRS_RST
                                    ; 416
                                    ; 417   ;Read the TX status register to clear any pending interrupts.
U 0020, 0212,0147,7777,7775,7760,3400   ; 418           NI_TX_STATUS_READ
                                    ; 419
                                    ; 420   ;Now set the register values derived from TWRAP, RWRAP, and MSIZE.
                                    ; 421
                                    ; 422   ;RWBASE := TWRAP
U 0021, 0222,0334,3527,7777,7760,3400   ; 423           DEBUG,ALUA/TWRAP,ALUB/RWBASE,ALUFN/A,ALUDEST/BRAM
                                    ; 424   ;RDBASE := TWRAP
U 0022, 0232,0334,4127,7777,7760,3400   ; 425           DEBUG,ALUA/TWRAP,ALUB/RDBASE,ALUFN/A,ALUDEST/BRAM
                                    ; 426   ;RWCUR := TWRAP + 1
U 0023, 0242,1304,4527,7777,7760,3400   ; 427           DEBUG,ALUA/TWRAP,ALUB/RWCUR,ALUFN/ADDZA,ALUDEST/BRAM,CRY_1
                                    ; 428   ;RDCUR := TWRAP + 1
U 0024, 0252,1304,5127,7777,7760,3400   ; 429           DEBUG,ALUA/TWRAP,ALUB/RDCUR,ALUFN/ADDZA,ALUDEST/BRAM,CRY_1
                                    ; 430   ;RDLIMIT := RWRAP - 1
U 0025, 0262,0314,6707,7777,7760,3400   ; 431           DEBUG,ALUA/RWRAP,ALUB/RDLIMIT,ALUFN/SUBZA,ALUDEST/BRAM,CRY_0
                                    ; 432   ;Compute TWRAP-MSIZE-1 and put result in Q-reg
U 0026, 0272,0021,7127,7777,7760,3400   ; 433           DEBUG,ALUA/TWRAP,ALUB/MSIZE,ALUFN/SUBAB,ALUDEST/Q,CRY_0
                                    ; 434   ;TBLEFT := TWRAP - MSIZE - 1
U 0027, 0302,0332,3367,7777,7760,3400   ; 435           DEBUG,ALUA/TEMP,ALUB/TBLEFT,ALUFN/Q,ALUDEST/BRAM
                                    ; 436   ;Jump to the dispatch vector
U 0030, 3406,0147,7777,7777,7760,3400   ; 437           JUMP_VECTOR
                                    ; 438
                                    ; 439
```

```
                                        ; 440   .TOC "Idle Routine"
                                        ; 441
                                        ; 442   ;IDLE - Idle routine.
                                        ; 443   ;--------------------
                                        ; 444   ; 2 if NI running
                                        ; 445   ; 5 if nothing to do
                                        ; 446   ; 8 to 12 to start a transfer (10 is standard case - no wraps, etc.).
                                        ; 447   ; 4 or 6 to process a TX done interrupt.
                                        ; 448   ; 6 to turn off RX, 12 to turn on RX
                                        ; 449   ;
                                        ; 450   ;Note that for the TX buffer, TWCUR and TWBASE are initially the same.
                                        ; 451   ;
                                        ; 452   ;       RUNNING.ATN_L
                                        ; 453   ;         0      0      No TX.  Check interrupt.
                                        ; 454   ;         0      1      No TX in progress, no interrupt.  Maybe start xfer.
                                        ; 455   ;         1      0      TX in progress, do nothing.
                                        ; 456   ;         1      1      TX in progress, do nothing.
                                        ; 457
                                        ; 458   ;Branch on NI running or not.  Do TWBASE-TDBASE for empty check.
                                        ; 459   ;Warning: this instruction is repeated in the vector table!
                                        ; 460   IDLE:   J/IDLE.A,COND/RUNNING.ATN_L,DEBUG,ALUA/TWBASE,ALUB/TDBASE,ALUFN/SUBAB,
U 0031, 0341,1121,0407,7777,7760,3400   ; 461            ALUDEST/NOP,CRY_1
                                        ; 462   =00
                                        ; 463   ;0.0 - NI is off.  Read TX status register into TEMP, join TX_ACQ_ERR checks.
U 0034, 0602,0337,7777,7775,7760,3400   ; 464   IDLE.A: J/INT.A,NI_TX_STATUS_READ,ALUA/TEMP,ALUB/TEMP,ALUFN/D,ALUDEST/BRAM
                                        ; 465
                                        ; 466   ;0.1 -  NI is off, no interrupts. Branch on empty check. Put TWBASE on bus
                                        ; 467   ; and increment TWCUR (note that TWCUR = TWBASE right now).
                                        ; 468           J/IDLE.B,COND/ZERO,DALU_READ,ALUA/TWCUR,ALUB/TWCUR,ALUFN/ADDZA,
U 0035, 0320,1204,1047,7777,7760,3400   ; 469            ALUDEST/BRAMA,CRY_1
                                        ; 470
                                        ; 471   ;1.0 - We are already transmitting.  Do nothing.
U 0036, 3406,0147,7777,7777,7760,3400   ; 472           JUMP_VECTOR
                                        ; 473
                                        ; 474   ;1.1 - We are already transmitting.  Do nothing.
U 0037, 3406,0147,7777,7777,7760,3400   ; 475           JUMP_VECTOR
                                        ; 476   =
                                        ; 477   =0
                                        ; 478   ;Not empty.  Do TWTOP := BUFFER[TWBASE] (may be positive or negative).
U 0032, 0442,0337,2115,7777,7760,3400   ; 479   IDLE.B: J/IDLE.E,BUFR_READ,ALUA/TWTOP,ALUB/TWTOP,ALUFN/D,ALUDEST/BRAM
                                        ; 480
                                        ; 481   ;Transmit and Deposit base pointers point to same place.  We are either empty
                                        ; 482   ; or are totally full.  Go check TX_AVAIL_L to decide which, put TWBASE on bus.
                                        ; 483   ; Note that the double COND is a crock to get around a lack of wires.  We
                                        ; 484   ; ignore MBUS_VALID and just duplicate the code handling TX_AVAIL_L.
                                        ; 485           J/IDLE.D,COND/TX_AVAIL_L.MBUS_VALID,DALU_READ,ALUA/TWBASE,ALUB/TWBASE,
U 0033, 0401,4134,0007,7777,7760,3400   ; 486            ALUFN/A,ALUDEST/NOP
                                        ; 487   =
                                        ; 488
```

```
                                    ; 489
                                    ; 490   ;IDLE - Start a transmission.
                                    ; 491   ;----------------------------
                                    ; 492
                                    ; 493   =00
                                    ; 494   ;Withdrawl and deposit point to same spot and we have room, i.e. we are empty.
                                    ; 495   ; Go see if someone wants to toggle packet reception.  First decrement TWCUR.
                                    ; 496   IDLE.D: J/RCV.A,COND/RX_ENABLE_OUT.IN,DEBUG,ALUA/TWCUR,ALUB/TWCUR,ALUFN/SUBZA,
U 0040, 0700,4314,1047,7777,7760,3400   ; 497            ALUDEST/BRAM,CRY_0
                                    ; 498
                                    ; 499   ;(Duplicate)
                                    ; 500           J/RCV.A,COND/RX_ENABLE_OUT.IN,DEBUG,ALUA/TWCUR,ALUB/TWCUR,ALUFN/SUBZA,
U 0041, 0700,4314,1047,7777,7760,3400   ; 501            ALUDEST/BRAM,CRY_0
                                    ; 502
                                    ; 503   ;The rare case of the TX buffer being totally full.
                                    ; 504   ;Do TWTOP := BUFFER[TWBASE] (may be positive or negative).
U 0042, 0442,0337,2115,7777,7760,3400   ; 505           J/IDLE.E,BUFR_READ,ALUA/TWTOP,ALUB/TWTOP,ALUFN/D,ALUDEST/BRAM
                                    ; 506
                                    ; 507   ;(Duplicate)
U 0043, 0442,0337,2115,7777,7760,3400   ; 508           J/IDLE.E,BUFR_READ,ALUA/TWTOP,ALUB/TWTOP,ALUFN/D,ALUDEST/BRAM
                                    ; 509   =
                                    ; 510
                                    ; 511   ;We are definitely going to start transmitting.  Set TX_REQ_ENABLE, branch
                                    ; 512   ; on whether the length we read from the buffer was negative, and compute
                                    ; 513   ; TWRAP-TWCUR to see if we need to wrap TWCUR.
                                    ; 514   IDLE.E: J/IDLE.F,COND/NEG,DEBUG,SET_STATUS,TX_REQ_ENABLE,ALUA/TWRAP,ALUB/TWCUR,
U 0044, 0460,3121,1127,7777,7730,3400   ; 515            ALUFN/SUBAB,ALUDEST/NOP,CRY_1
                                    ; 516   =0
                                    ; 517   ;The length was positive.  Do TWTOP := TWBASE + <length>.  Branch on wrap.
                                    ; 518   IDLE.F: J/IDLE.G,COND/ZERO,DEBUG,ALUA/TWBASE,ALUB/TWTOP,ALUFN/ADDAB,
U 0046, 0500,0301,2007,7777,7760,3400   ; 519           ALUDEST/BRAM
                                    ; 520
                                    ; 521   ;The length was negative.  Do TWTOP := TWBASE - <- length>.  Branch on wrap.
                                    ; 522           J/IDLE.G,COND/ZERO,DEBUG,ALUA/TWBASE,ALUB/TWTOP,ALUFN/SUBAB,
U 0047, 0500,1321,2007,7777,7760,3400   ; 523            ALUDEST/BRAM,CRY_1
                                    ; 524   =
                                    ; 525
                                    ; 526   =0
                                    ; 527   ;No wrap needed. Mod TWTOP by TWRAP.  Turn off TX_EOF
                                    ; 528   IDLE.G: J/IDLE.H,DEBUG,CLEAR_STATUS,TX_EOF,ALUA/TWRAP,ALUB/TWTOP,ALUFN/NAANDB,
U 0050, 0452,0351,2127,7777,7660,3400   ; 529             ALUDEST/BRAM
                                    ; 530
                                    ; 531   ;Here to wrap TWCUR.  Turn off TX_EOF
                                    ; 532           J/IDLE.K,DEBUG,CLEAR_STATUS,TX_EOF,ALUA/TWCUR,ALUB/TWCUR,ALUFN/Z,
U 0051, 0562,0347,1047,7777,7660,3400   ; 533             ALUDEST/BRAM
                                    ; 534   =
                                    ; 535
```

```
                                    ; 536
                                    ; 537   ;IDLE - Start a transmission
                                    ; 538   ;---------------------------
                                    ; 539
                                    ; 540   ;This code does the extra increment of TWCUR (see monument). We start at
                                    ; 541   ; IDLE.H if we did not wrap previously, IDLE.K otherwise.
                                    ; 542
                                    ; 543   ;Increment TWCUR
U 0045, 0522,1304,1047,7777,7760,3400   ; 544   IDLE.H: DEBUG,ALUA/TWCUR,ALUB/TWCUR,ALUFN/ADDZA,ALUDEST/BRAM,CRY_1
                                    ; 545
                                    ; 546   ;Check for wrap by doing TWCUR-TWRAP
U 0052, 0532,1121,2447,7777,7760,3400   ; 547           DEBUG,ALUA/TWCUR,ALUB/TWRAP,ALUFN/SUBAB,ALUDEST/NOP,CRY_1
                                    ; 548
                                    ; 549   ;See if we need to wrap
U 0053, 0540,0147,7777,7777,7760,3400   ; 550           J/IDLE.J,COND/ZERO
                                    ; 551
                                    ; 552   =0
                                    ; 553   ;No wrap necessary, just vector
U 0054, 3406,0147,7777,7777,7760,3400   ; 554   IDLE.J: JUMP_VECTOR
                                    ; 555
                                    ; 556   ;Wrap TWCUR to zero and then vector
U 0055, 3406,0347,1047,7777,7760,3400   ; 557           JUMP_VECTOR,DEBUG,ALUA/TWCUR,ALUB/TWCUR,ALUFN/Z,ALUDEST/BRAM
                                    ; 558   =
                                    ; 559
                                    ; 560   ;Here after we wrapped because of incrementing TWCUR.  Do TWTOP mod TWRAP.
U 0056, 0572,0351,2127,7777,7760,3400   ; 561   IDLE.K: DEBUG,ALUA/TWRAP,ALUB/TWTOP,ALUFN/NAANDB,ALUDEST/BRAM
                                    ; 562
                                    ; 563   ;Increment TWCUR and vector (already at 0, so no need to check for wrap).
U 0057, 3406,0304,1047,7777,7760,3400   ; 564           JUMP_VECTOR,DEBUG,ALUA/TWCUR,ALUB/TWCUR,ALUFN/ADDZA,ALUDEST/BRAM
                                    ; 565
```

```
                                        ; 566
                                        ; 567   ;IDLE - Check for transmission errors.
                                        ; 568   ;-------------------------------------
                                        ; 569
                                        ; 570   ;This code checks for transmission errors.  The defined bits (from right to
                                        ; 571   ; left) are listed below.  We check for bits 0 or 2 being set and set
                                        ; 572   ; TX_ACQ_ERR if either flag is up.
                                        ; 573   ;
                                        ; 574   ;               0 - transmit underflow
                                        ; 575   ;               1 - transmit collision
                                        ; 576   ;               2 - transmit timeout
                                        ; 577   ;               3 - transmit okay
                                        ; 578   ;               7 - old/new status
                                        ; 579
                                        ; 580   ;See if bit 0 was set (TX underflow) and shift down once
                                        ; 581   INT.A:  J/INT.B,COND/DBUS_LSB,DALU_READ,ALUA/TEMP,ALUB/TEMP,ALUFN/A,
U 0060, 0620,6534,7767,7777,7760,3400   ; 582            ALUDEST/BDOWN
                                        ; 583   =0
                                        ; 584   ;The LSB was zero, so no problem.  Shift down again
U 0062, 0612,0534,7767,7777,7760,3400   ; 585   INT.B:  J/INT.C,DALU_READ,ALUA/TEMP,ALUB/TEMP,ALUFN/A,ALUDEST/BDOWN
                                        ; 586
                                        ; 587   ;The LSB is set, call the lossage an acquisition error, then vector
U 0063, 3406,0147,7777,7777,3770,3400   ; 588           JUMP_VECTOR,SET_STATUS,TX_ACQ_ERR
                                        ; 589   =
                                        ; 590
                                        ; 591   ;See if bit 2 was set (TX timeout).
U 0061, 0640,6147,7777,7777,7760,3400   ; 592   INT.C:  J/INT.D,COND/DBUS_LSB
                                        ; 593
                                        ; 594   =0
                                        ; 595   ;The LSB is zero, no problems.  Just vector
U 0064, 3406,0147,7777,7777,7760,3400   ; 596   INT.D:  JUMP_VECTOR
                                        ; 597
                                        ; 598   ;The LSB is set, call the lossage an acquisition error, then vector
U 0065, 3406,0147,7777,7777,3770,3400   ; 599           JUMP_VECTOR,SET_STATUS,TX_ACQ_ERR
                                        ; 600   =
                                        ; 601
```

```
                                    ; 602
                                    ; 603   ;IDLE - Reception toggling.
                                    ; 604   ;-------------------------
                                    ; 605   ;
                                    ; 606   ;Here to toggle packet reception.  Last instruction was at IDLE.D.
                                    ; 607   ; Note that it doesn't matter if we take a lot of instructions to toggle
                                    ; 608   ; packet reception since either nothing has been happening or nothing will
                                    ; 609   ; be happening.
                                    ; 610
                                    ; 611   =00
                                    ; 612   ;0.0 - do nothing
U 0070, 3406,0147,7777,7777,7760,3400  ; 613   RCV.A:  JUMP_VECTOR
                                    ; 614
                                    ; 615   ;0.1 - enable reception (write a 100 to the NI command register)
                                    ; 616   ;First we create a zero in TEMP.  Also reset bad address line.
U 0071, 0662,0347,7777,7577,7760,3400  ; 617           J/RCV.C,NI_BADRS_RST,ALUA/TEMP,ALUB/TEMP,ALUFN/Z,ALUDEST/BRAM
                                    ; 618
                                    ; 619   ;1.0 - disable reception (write a zero to the NI command register)
                                    ; 620   ;We drop our status line and write a zero the NI.  We let the status line
                                    ; 621   ; stabilize before jumping to the vector routine (waste a cycle).
                                    ; 622           J/RCV.B,CLEAR_STATUS,RX_ENABLE,NI_RX_COMMAND_LOAD,DALU_READ,ALUA/TEMP,
U 0072, 0732,0147,7767,7776,7740,3000  ; 623            ALUB/TEMP,ALUFN/Z,ALUDEST/NOP
                                    ; 624
                                    ; 625   ;1.1 - do nothing
U 0073, 3406,0147,7777,7777,7760,3400  ; 626   RCV.B:  JUMP_VECTOR
                                    ; 627   =
                                    ; 628
                                    ; 629   ;Add a 1 to 0 and shift up to get a 2.
U 0066, 0672,1704,7777,7777,7760,3400  ; 630   RCV.C:  ALUA/TEMP,ALUB/TEMP,ALUFN/ADDZA,ALUDEST/BUP,CRY_1
                                    ; 631   ;Shift 2 up to 4
U 0067, 0742,0734,7777,7777,7760,3400  ; 632           ALUA/TEMP,ALUB/TEMP,ALUFN/A,ALUDEST/BUP
                                    ; 633   ;Shift 4 up to 10
U 0074, 0752,0734,7777,7777,7760,3400  ; 634           ALUA/TEMP,ALUB/TEMP,ALUFN/A,ALUDEST/BUP
                                    ; 635   ;Shift 10 up to 20
U 0075, 0762,0734,7777,7777,7760,3400  ; 636           ALUA/TEMP,ALUB/TEMP,ALUFN/A,ALUDEST/BUP
                                    ; 637   ;Shift 20 up to 40
U 0076, 0772,0734,7777,7777,7760,3400  ; 638           ALUA/TEMP,ALUB/TEMP,ALUFN/A,ALUDEST/BUP
                                    ; 639   ;Shift 40 up to 100.  Do the set RX_ENABLE one cycle before the vector
U 0077, 1002,0734,7777,7777,7750,3400  ; 640           SET_STATUS,RX_ENABLE,ALUA/TEMP,ALUB/TEMP,ALUFN/A,ALUDEST/BUP
                                    ; 641
                                    ; 642   ;Write 100 to the NI, then vector
                                    ; 643           JUMP_VECTOR,NI_RX_COMMAND_LOAD,DALU_READ,ALUA/TEMP,ALUB/TEMP,ALUFN/A,
U 0100, 3406,0134,7767,7776,7760,3000  ; 644            ALUDEST/NOP
                                    ; 645
                                    ; 646
```

```
                                        ; 647   .TOC "Vector Requests"
                                        ; 648
                                        ; 649   ;Vector Request Dispatch Table
                                        ; 650   ;---------------------------
                                        ; 651
                                        ; 652   =11100000
                                        ; 653   VECTOR:
                                        ; 654
                                        ; 655   ;Requests 0-7 are idle routine requests.  Only offset 7 is ever used.
                                        ; 656
U 0340, 0312,0147,7777,7777,7760,3400   ; 657           J/IDLE
U 0341, 0312,0147,7777,7777,7760,3400   ; 658           J/IDLE
U 0342, 0312,0147,7777,7777,7760,3400   ; 659           J/IDLE
U 0343, 0312,0147,7777,7777,7760,3400   ; 660           J/IDLE
U 0344, 0312,0147,7777,7777,7760,3400   ; 661           J/IDLE
U 0345, 0312,0147,7777,7777,7760,3400   ; 662           J/IDLE
U 0346, 0312,0147,7777,7777,7760,3400   ; 663           J/IDLE
                                        ; 664   ;07 - First instruction of the IDLE routine
                                        ; 665   ;Branch on NI running or not.  Do TWBASE-TDBASE for empty check.
                                        ; 666           J/IDLE.A,COND/RUNNING.ATN_L,DEBUG,ALUA/TWBASE,ALUB/TDBASE,ALUFN/SUBAB,
U 0347, 0341,1121,0407,7777,7760,3400   ; 667            ALUDEST/NOP,CRY_1
                                        ; 668
                                        ; 669   ;10 - Xport read request
U 0350, 3604,0147,7777,7777,7760,3400   ; 670           J/R.FIELD,COND/RCON_REQ
                                        ; 671
                                        ; 672   ;11 - Xport load request
U 0351, 3604,0147,7777,7777,7760,3400   ; 673           J/R.FIELD,COND/RCON_REQ
                                        ; 674
                                        ; 675   ;12 - TX to NI
                                        ; 676   ;Compute TWCUR-TWTOP to see if we're at the last data word
U 0352, 1012,1121,2057,7777,7760,3400   ; 677           J/V12,ALUA/TWCUR,ALUB/TWTOP,ALUFN/SUBAB,ALUDEST/NOP,CRY_1
                                        ; 678
                                        ; 679   ;13 - NI to RX
                                        ; 680   ;Load RDCUR onto DBus, increment RDCUR
U 0353, 1172,1204,5247,7777,7760,3400   ; 681           J/V13,DALU_READ,ALUA/RDCUR,ALUB/RDCUR,ALUFN/ADDZA,ALUDEST/BRAMA,CRY_1
                                        ; 682
                                        ; 683   ;14 - RX abort.
                                        ; 684   ;Jump to common abort code after preparing RDCUR wrap check (RWRAP-(RDBASE+1))
U 0354, 2352,0121,4317,7777,7760,3400   ; 685           J/V13ABT,ALUA/RWRAP,ALUB/RDBASE,ALUFN/SUBAB,ALUDEST/NOP
                                        ; 686
                                        ; 687   ;15 - TX abort.
                                        ; 688   ;Shut off NI and do TWCUR := TWBASE.
                                        ; 689           J/V15,CLEAR_STATUS,TX_REQ_ENABLE,ALUA/TWBASE,ALUB/TWCUR,ALUFN/A,
U 0355, 2402,0334,1017,7777,7720,3400   ; 690            ALUDEST/BRAM
                                        ; 691
                                        ; 692   ;16 - Mport Load.  RX to Barrel.
                                        ; 693   ;Do RWCUR-TWTOP to see if we already sent last data word
U 0356, 2412,1121,5637,7777,7760,3400   ; 694           J/V16,ALUA/RWCUR,ALUB/RWTOP,ALUFN/SUBAB,ALUDEST/NOP,CRY_1
                                        ; 695
                                        ; 696   ;17 - Mport Read.  Barrel to TX.
                                        ; 697   ;Increment TDCUR and load address onto data bus
U 0357, 2452,1203,1467,7777,7760,3400   ; 698           J/V17,DALU_READ,ALUA/TDCUR,ALUB/TDCUR,ALUFN/ADDZB,ALUDEST/BRAMA,CRY_1
                                        ; 699   =
                                        ; 700
```

```
                                    ; 701
                                    ; 702   ;Vector Request 12 - transmit buffer to Ethernet
                                    ; 703   ;-----------------------------------------------
                                    ; 704   ; Not at EOF 5 ins; at EOF 10 or 11 ins
                                    ; 705
                                    ; 706   ;Compute TWCUR-TWTOP to see if we're at the last data word
                                    ; 707   ;V..12: J/V12,ALUA/TWCUR,ALUB/TWTOP,ALUFN/SUBAB,ALUDEST/NOP,CRY_1
                                    ; 708
                                    ; 709   ;We test the previous result, put TWCUR on data bus, and increment TWCUR
                                    ; 710   ; We also set low byte valid (we may change our mind if at EOF).
                                    ; 711   V12:    J/V12A,COND/ZERO,SET_STATUS,LBYTE_VALID,DALU_READ,ALUA/TWCUR,
U 0101, 1020,1203,1047,7777,7570,3400   ; 712            ALUB/TWCUR,ALUFN/ADDZB,ALUDEST/BRAMA,CRY_1
                                    ; 713   =0
                                    ; 714   ;Not at EOF. Write data word to the NI and compute TWCUR-TWRAP.  Be careful
                                    ; 715   ; not to put anything on the Dbus for one more cycle.
                                    ; 716   V12A:   J/V12B,NI_DATA_LOAD,BUFR_READ,ALUA/TWCUR,ALUB/TWRAP,ALUFN/SUBAB,
U 0102, 1042,1121,2455,7773,7760,3400   ; 717            ALUDEST/NOP,CRY_1
                                    ; 718
                                    ; 719   ;Here on last word.  We set TX_EOF and load TWBASE onto data bus since
                                    ; 720   ; we cannot send the last word right away.
                                    ; 721           J/V12D,DALU_READ,SET_STATUS,TX_EOF,ALUA/TWBASE,ALUB/TWBASE,ALUFN/A,
U 0103, 1052,0134,0007,7777,7670,3400   ; 722            ALUDEST/NOP
                                    ; 723   =
                                    ; 724   ;Here on a short call to branch after computing TWCUR-TWRAP
U 0104, 1060,0147,7777,7777,7760,3400   ; 725   V12B:   J/V12C,COND/ZERO
                                    ; 726
                                    ; 727   =0
                                    ; 728   ;Here on short write to do nothing except return
U 0106, 3406,0147,7777,7777,7760,3400   ; 729   V12C:   JUMP_VECTOR
                                    ; 730
                                    ; 731   ;Here on short write to wrap TWCUR and return
U 0107, 3406,0347,1377,7777,7760,3400   ; 732           JUMP_VECTOR,ALUA/TEMP,ALUB/TWCUR,ALUFN/Z,ALUDEST/BRAM
                                    ; 733   =
                                    ; 734
```

```
                                    ; 735
                                    ; 736   ;Vector Request 12 - transmit buffer to Ethernet (cont'd)
                                    ; 737   ;-------------------------------------------------------
                                    ; 738
                                    ; 739   ;Here on End of Frame.
                                    ; 740
                                    ; 741   ;Read length from buffer into TEMP.
U 0105, 1102,0337,7775,7777,7760,3400   ; 742   V12D:   BUFR_READ,ALUA/TEMP,ALUB/TEMP,ALUFN/D,ALUDEST/BRAM
                                    ; 743
                                    ; 744   ;Branch on length negative (low byte invalid).  Put negative length in Q
U 0110, 1120,3024,7777,7777,7760,3400   ; 745           J/V12E,COND/NEG,ALUA/TEMP,ALUB/TEMP,ALUFN/SUBAZ,ALUDEST/Q,CRY_1
                                    ; 746
                                    ; 747   =0
                                    ; 748   ;Compute TBLEFT := TBLEFT+BUFFER[TWBASE]+1 (words available in TX buffer)
U 0112, 1112,1301,3377,7777,7760,3400   ; 749   V12E:   J/V12F,ALUA/TEMP,ALUB/TBLEFT,ALUFN/ADDAB,ALUDEST/BRAM,CRY_1
                                    ; 750
                                    ; 751   ;Compute words left in TX buffer as above. Set low byte invalid.
                                    ; 752           J/V12F,CLEAR_STATUS,LBYTE_VALID,ALUA/TBLEFT,ALUB/TBLEFT,ALUFN/ADDAQ,
U 0113, 1112,1300,3157,7777,7560,3400   ; 753            ALUDEST/BRAM,CRY_1
                                    ; 754   =
                                    ; 755   ;Branch on previous result negative.  If so, then we don't have any room in
                                    ; 756   ; the TX buffer and TX_BUFR_AVAIL_L should be set (it is active low).
                                    ; 757   ; Load TWCUR-1 onto data bus (that pointer was already incremented!).
                                    ; 758   V12F:   J/V12G,COND/NEG,DALU_READ,ALUA/TWCUR,ALUB/TWCUR,ALUFN/SUBZA,
U 0111, 1140,2114,1047,7777,7760,3400   ; 759            ALUDEST/NOP,CRY_0
                                    ; 760   =0
                                    ; 761   ;Load the last data word and compute TWRAP-TWCUR.  Also say we have room.
                                    ; 762   ; Be careful not to put anything on the Dbus for at least one more cycle.
                                    ; 763   V12G:   J/V12H,NI_DATA_LOAD,BUFR_READ,CLEAR_STATUS,TX_BUFR_AVAIL_L,ALUA/TWCUR,
U 0114, 1162,1121,2455,7773,6760,3400   ; 764            ALUB/TWRAP,ALUFN/SUBAB,ALUDEST/NOP,CRY_1
                                    ; 765
                                    ; 766   ;Load the last data word and compute TWRAP-TWCUR.  Say we have no room.
                                    ; 767   ; Be careful not to put anything on the Dbus for at least one more cycle.
                                    ; 768           J/V12H,NI_DATA_LOAD,BUFR_READ,SET_STATUS,TX_BUFR_AVAIL_L,ALUA/TWCUR,
U 0115, 1162,1121,2455,7773,6770,3400   ; 769            ALUB/TWRAP,ALUFN/SUBAB,ALUDEST/NOP,CRY_1
                                    ; 770   =
                                    ; 771   ;Branch on wrap.  Shut off the NI.
U 0116, 1200,0147,7777,7777,7720,3400   ; 772   V12H:   J/V12J,COND/ZERO,CLEAR_STATUS,TX_REQ_ENABLE
                                    ; 773
                                    ; 774   =0
                                    ; 775   ;Do TWBASE:=TWCUR, and return to vector routine
U 0120, 3406,0334,0057,7777,7760,3400   ; 776   V12J:   JUMP_VECTOR,ALUA/TWCUR,ALUB/TWBASE,ALUFN/A,ALUDEST/BRAM
                                    ; 777
                                    ; 778   ;Here to wrap TWCUR
U 0121, 1202,0347,1057,7777,7760,3400   ; 779           J/V12J,ALUA/TWCUR,ALUB/TWCUR,ALUFN/Z,ALUDEST/BRAM
                                    ; 780   =
                                    ; 781
```

```
                                        ; 782
                                        ; 783   ;Vector Request 13 - Ethernet to receive buffer
                                        ; 784   ;-----------------
                                        ; 785   ;Not at EOF:    overflow or bad address - 4 ins
                                        ; 786   ;               Normal - 5 or 6 ins
                                        ; 787   ;If at EOF:     If flush - 7 ins
                                        ; 788   ;               If ends at RWRAP:  okay - 10 ins (11 if LByte invalid)
                                        ; 789   ;                                  abort - 10 or 12 ins
                                        ; 790   ;               Standard case:     okay - 12 ins
                                        ; 791   ;                                  abort - 8 or 12 ins
                                        ; 792
                                        ; 793   ; Read data from the NI, possibly handle End of Frame and Low Byte Valid.
                                        ; 794
                                        ; 795   ; Truth table for first COND.  Note well: both conditions are active low!
                                        ; 796   ;       RX_EOF.LBYTE
                                        ; 797   ;         0      0      EOF, low byte is valid
                                        ; 798   ;         0      1      EOF, low byte not valid
                                        ; 799   ;         1      0      not EOF, don't care about low byte (should be valid)
                                        ; 800   ;         1      1      not EOF, don't care about low byte (should be valid)
                                        ; 801
                                        ; 802   ;Load RDCUR onto DBus, increment RDCUR
                                        ; 803   ;V..13: J/V13,DALU_READ,ALUA/RDCUR,ALUB/RDCUR,ALUFN/ADDZA,ALUDEST/BRAMA,CRY_1
                                        ; 804
                                        ; 805   ;Write from NI to buffer, compute RDCUR-RWRAP (for wrap check).  Do a four-way
                                        ; 806   ; branch on End of Frame and Low Byte Valid.
                                        ; 807   V13:    J/V13A,COND/RX_EOF.LBYTE,NI_DATA_READ,BUFR_LOAD,ALUA/RDCUR,
U 0117, 1241,7121,6256,7767,7760,3400   ; 808            ALUB/RWRAP,ALUFN/SUBAB,ALUDEST/NOP,CRY_1
                                        ; 809
                                        ; 810   =00
                                        ; 811   ;0.0 - EOF and LByte valid.
                                        ; 812   ;Compute four-way branch on wrap and packet flush conditions.
                                        ; 813   ; Do RDCUR-RDBASE-1 to Q-reg just in case.
U 0124, 1401,2021,4257,7777,7760,3400   ; 814   V13A:   J/V13EOF,COND/ZERO.FLUSH,ALUA/RDCUR,ALUB/RDBASE,ALUFN/SUBAB,ALUDEST/Q
                                        ; 815
                                        ; 816   ;0.1 - EOF and LByte invalid
                                        ; 817   ;Compute four-way branch on wrap and packet flush conditions.
                                        ; 818   ; Do RDCUR-RDBASE-1 to Q-reg just in case.
U 0125, 1441,2021,4257,7777,7760,3400   ; 819           J/V13EOB,COND/ZERO.FLUSH,ALUA/RDCUR,ALUB/RDBASE,ALUFN/SUBAB,ALUDEST/Q
                                        ; 820
                                        ; 821   ;1.0 - Not at EOF, don't care about LByte
                                        ; 822   ;Compute four-way branch on wrap and packet overflow and do RDCUR-RDLIMIT in
                                        ; 823   ; case we need to check for overflow.
                                        ; 824           J/V13NRM,COND/ZERO.FLUSH,ALUA/RDCUR,ALUB/RDLIMIT,ALUFN/SUBAB,
U 0126, 1301,3121,6657,7777,7760,3400   ; 825            ALUDEST/NOP,CRY_1
                                        ; 826
                                        ; 827   ;1.1 - Not at EOF, don't care about LByte
                                        ; 828   ;Compute four-way branch on wrap and packet overflow and do RDCUR-RDLIMIT in
                                        ; 829   ; case we need to check for overflow.
                                        ; 830           J/V13NRM,COND/ZERO.FLUSH,ALUA/RDCUR,ALUB/RDLIMIT,ALUFN/SUBAB,
U 0127, 1301,3121,6657,7777,7760,3400   ; 831            ALUDEST/NOP,CRY_1
                                        ; 832   =
                                        ; 833
```

```
                                    ; 834
                                    ; 835   ;Vector Request 13 - Not at End of Frame.
                                    ; 836   ;---------------------------------------
                                    ; 837   ;
                                    ; 838   ;We are handling the normal case.  We just did a four-way branch on last ALU
                                    ; 839   ; result was zero (wrap RDCUR) and RX_OVERFLOW.  The circuitry for that COND
                                    ; 840   ; is the same as for the EOF case.
                                    ; 841   ;
                                    ; 842   ;       ZERO.FLUSH
                                    ; 843   ;         0   0         No wrap, no overflow.  Must check for overflow
                                    ; 844   ;         0   1         Overflow.  Decrement RDCUR and return.
                                    ; 845   ;         1   0         Wrap, no overflow.  Must check for overflow.
                                    ; 846   ;         1   1         Overflow.  Decrement RDCUR and return.
                                    ; 847
                                    ; 848   =00
                                    ; 849   ;0.0 - no wrap, no known overflow.  Branch on result of overflow check.
U 0130, 1220,0147,7777,7777,7760,3400  ; 850   V13NRM: J/V13NRA,COND/ZERO
                                    ; 851
                                    ; 852   ;0.1 - overflow. Decrement RDCUR and vector.
U 0131, 3406,0314,5257,7777,7760,3400  ; 853           JUMP_VECTOR,ALUA/RDCUR,ALUB/RDCUR,ALUFN/SUBZA,ALUDEST/BRAM,CRY_0
                                    ; 854
                                    ; 855   ;1.0 - wrap and check for overflow.  Do TWRAP-RDLIMIT here.  Set RDCUR later.
U 0132, 1342,1121,6537,7777,7760,3400  ; 856           J/V13NRB,ALUA/TWRAP,ALUB/RDLIMIT,ALUFN/SUBAB,ALUDEST/NOP,CRY_1
                                    ; 857
                                    ; 858   ;1.1 - overflow. Decrement RDCUR and vector.
U 0133, 3406,0314,5257,7777,7760,3400  ; 859           JUMP_VECTOR,ALUA/RDCUR,ALUB/RDCUR,ALUFN/SUBZA,ALUDEST/BRAM,CRY_0
                                    ; 860   =
                                    ; 861   =0
                                    ; 862   ;No overflow, just vector
U 0122, 3406,0147,7777,7777,7760,3400  ; 863   V13NRA: JUMP_VECTOR
                                    ; 864
                                    ; 865   ;We just detected overflow.  Decrement RDCUR, set RX_OVERFLOW, and vector
                                    ; 866           JUMP_VECTOR,SET_STATUS,RX_OVERFLOW,ALUA/RDCUR,ALUB/RDCUR,ALUFN/SUBZA,
U 0123, 3406,0314,5257,7777,5770,3400  ; 867            ALUDEST/BRAM,CRY_0
                                    ; 868   =
                                    ; 869
                                    ; 870   ;We now check for overflow.  Still need to set RDCUR to TWRAP if no overflow
U 0134, 1360,0147,7777,7777,7760,3400  ; 871   V13NRB: J/V13NRC,COND/ZERO
                                    ; 872
                                    ; 873   =0
                                    ; 874   ;Wrapped and no overflow.  Set RDCUR to TWRAP and vector
U 0136, 3406,0334,5137,7777,7760,3400  ; 875   V13NRC: JUMP_VECTOR,ALUA/TWRAP,ALUB/RDCUR,ALUFN/A,ALUDEST/BRAM
                                    ; 876
                                    ; 877   ;Overflow at wrap point. Decrement RDCUR (wasn't changed!), set RX_OVERFLOW,
                                    ; 878   ; and vector.
                                    ; 879           JUMP_VECTOR,SET_STATUS,RX_OVERFLOW,ALUA/RDCUR,ALUB/RDCUR,ALUFN/SUBZA,
U 0137, 3406,0314,5257,7777,5770,3400  ; 880            ALUDEST/BRAM,CRY_0
                                    ; 881   =
                                    ; 882
                                    ; 883
```

```
                                        ; 884
                                        ; 885   ;Vector Request 13 - End of Frame switch yard.
                                        ; 886   ;-----------------------------------------
                                        ; 887   ;
                                        ; 888   ;We are at End of Frame (EOF).  We just did a four-way branch on last ALU
                                        ; 889   ; result was zero (wrap RDCUR) and the OR of RX_OVERFLOW and RX_BAD_ADDRESS.
                                        ; 890   ;
                                        ; 891   ;       ZERO.FLUSH
                                        ; 892   ;         0   0         No wrap. Q-reg & RDCUR valid.  May need 2nd wrap.
                                        ; 893   ;         0   1         Flush packet. Reset flags.
                                        ; 894   ;         1   0         Must wrap. Q-reg & RDCUR invalid. Never do 2nd wrap.
                                        ; 895   ;         1   1         Flush packet. Reset flags.
                                        ; 896
                                        ; 897   ;V13EOF handles the low byte valid condition.  Do a bad address reset if
                                        ; 898   ; the packet is valid -- resets all address mask state machines.
                                        ; 899
                                        ; 900   =00
                                        ; 901   ;0.0 - No wrap needed yet. Compute RDLIMIT-RDCUR for first overflow check
                                        ; 902   V13EOF: J/V13NWR,NI_BADRS_RST,ALUA/RDLIMIT,ALUB/RDCUR,ALUFN/SUBAB,ALUDEST/NOP,
U 0140, 1572,1121,5337,7577,7760,3400   ; 903            CRY_1
                                        ; 904
                                        ; 905   ;0.1-  Flush packet.  Prepare RDCUR wrap check and join common abort code.
U 0141, 2352,0121,4317,7777,7760,3400   ; 906           J/V13ABT,ALUA/RWRAP,ALUB/RDBASE,ALUFN/SUBAB,ALUDEST/NOP
                                        ; 907
                                        ; 908   ;1.0 - We are at the wrap point.  Packet ended at end of buffer. Put RDBASE
                                        ; 909   ;       onto the data bus and join the wrap code.
                                        ; 910           J/V13WRP,NI_BADRS_RST,DALU_READ,ALUA/RDBASE,ALUB/TEMP,ALUFN/A,
U 0142, 1512,0134,7607,7577,7760,3400   ; 911            ALUDEST/NOP
                                        ; 912
                                        ; 913   ;1.1 - Flush packet.  Prepare RDCUR wrap check and join common abort code.
U 0143, 2352,0121,4317,7777,7760,3400   ; 914           J/V13ABT,ALUA/RWRAP,ALUB/RDBASE,ALUFN/SUBAB,ALUDEST/NOP
                                        ; 915   =
                                        ; 916
                                        ; 917
                                        ; 918   ;V13EOB handles the low byte invalid condition.  Do a bad address reset if
                                        ; 919   ; the packet is valid -- resets all address mask state machines.
                                        ; 920
                                        ; 921   =00
                                        ; 922   ;0.0 - No wrap needed yet. Compute RDLIMIT-RDCUR for first overflow check
                                        ; 923   V13EOB: J/V3BNWR,NI_BADRS_RST,ALUA/RDLIMIT,ALUB/RDCUR,ALUFN/SUBAB,ALUDEST/NOP,
U 0144, 2072,1121,5337,7577,7760,3400   ; 924            CRY_1
                                        ; 925
                                        ; 926   ;0.1-  Flush packet.  Prepare RDCUR wrap check and join common abort code.
U 0145, 2352,0121,4317,7777,7760,3400   ; 927           J/V13ABT,ALUA/RWRAP,ALUB/RDBASE,ALUFN/SUBAB,ALUDEST/NOP
                                        ; 928
                                        ; 929   ;1.0 - We are at the wrap point.  Packet ended at end of buffer.
                                        ; 930   ;Do RWRAP-RDBASE-1 to Q-reg
                                        ; 931           J/V13WRB,NI_BADRS_RST,DALU_READ,ALUA/RDBASE,ALUB/RWRAP,ALUFN/SUBBA,
U 0146, 1352,0011,6207,7577,7760,3400   ; 932            ALUDEST/Q
                                        ; 933
                                        ; 934   ;1.1 - Flush packet.  Prepare RDCUR wrap check and join common abort code.
U 0147, 2352,0121,4317,7777,7760,3400   ; 935           J/V13ABT,ALUA/RWRAP,ALUB/RDBASE,ALUFN/SUBAB,ALUDEST/NOP
                                        ; 936   =
                                        ; 937
```

```
                                        ; 938
                                        ; 939   ;Vector Request 13 - EOF, packet ends at RWRAP.
                                        ; 940   ;-----------------------------------------------
                                        ; 941   ;
                                        ; 942   ;The code on this page handles EOF case 1.0, packet ends at RWRAP, no flush.
                                        ; 943   ;Since we are already at the wrap point, we needn't check for wrap again.
                                        ; 944   ;Entry points at V13WRB and V13WRP to handle low byte valid conditions.
                                        ; 945
                                        ; 946   ;Low byte is not valid.  Put RDBASE on Dbus.
U 0135, 1502,0134,4207,7777,7760,3400   ; 947   V13WRB: DALU_READ,ALUA/RDBASE,ALUB/RDBASE,ALUFN/A,ALUDEST/NOP
                                        ; 948
                                        ; 949   ;Write negative of length from Q-reg to buffer, then join common code.
                                        ; 950           J/V13WRA,BUFR_LOAD,DALU_READ,ALUA/TEMP,ALUB/TEMP,ALUFN/SUBZQ,
U 0150, 1522,1122,7766,7777,7760,3400   ; 951            ALUDEST/NOP,CRY_1
                                        ; 952
                                        ; 953
                                        ; 954   ;Low byte is valid.  Compute RWRAP-RDBASE-1 (pkt length) and write into buffer
U 0151, 1522,0121,4306,7777,7760,3400   ; 955   V13WRP: DALU_READ,BUFR_LOAD,ALUA/RWRAP,ALUB/RDBASE,ALUFN/SUBAB,ALUDEST/NOP
                                        ; 956
                                        ; 957   ;Do first overflow check (RDLIMIT-TWRAP)
U 0152, 1532,1121,2737,7777,7760,3400   ; 958   V13WRA: ALUA/RDLIMIT,ALUB/TWRAP,ALUFN/SUBAB,ALUDEST/NOP,CRY_1
                                        ; 959
                                        ; 960   ;Branch on first overflow check, do RDCUR := TWRAP + 1.
U 0153, 1540,1304,5137,7777,7760,3400   ; 961           J/V13WRC,COND/ZERO,ALUA/TWRAP,ALUB/RDCUR,ALUFN/ADDZA,ALUDEST/BRAM,CRY_1
                                        ; 962
                                        ; 963   =0
                                        ; 964   ;First overflow check okay. Do RDLIMIT-RDCUR for second overflow check.
U 0154, 1562,1121,5337,7777,7760,3400   ; 965   V13WRC: J/V13WRD,ALUA/RDLIMIT,ALUB/RDCUR,ALUFN/SUBAB,ALUDEST/NOP,CRY_1
                                        ; 966
                                        ; 967   ;We hit the deposit limit.  Jump to abort code after preparing RDCUR wrap chk
U 0155, 2352,0121,4317,7777,7760,3400   ; 968           J/V13ABT,ALUA/RWRAP,ALUB/RDBASE,ALUFN/SUBAB,ALUDEST/NOP
                                        ; 969   =
                                        ; 970
                                        ; 971   ;Branch on result of second overflow check.
U 0156, 1600,0147,7777,7777,7760,3400   ; 972   V13WRD: J/V13WRE,COND/ZERO
                                        ; 973
                                        ; 974   =0
                                        ; 975   ;No overflow.  Set new RDBASE, say packet is available, and vector.
                                        ; 976   V13WRE: JUMP_VECTOR,SET_STATUS,RX_PKT_AVAIL,ALUA/TWRAP,ALUB/RDBASE,ALUFN/A,
U 0160, 3406,0334,4137,7777,7370,3400   ; 977            ALUDEST/BRAM
                                        ; 978
                                        ; 979   ;We hit the deposit limit.  Jump to abort code after preparing RDCUR wrap chk
U 0161, 2352,0121,4317,7777,7760,3400   ; 980           J/V13ABT,ALUA/RWRAP,ALUB/RDBASE,ALUFN/SUBAB,ALUDEST/NOP
                                        ; 981   =
                                        ; 982
```

```
                                    ; 983
                                    ; 984   ;Vector Request 13 - EOF, Low byte valid
                                    ; 985   ;-------------------------------------
                                    ; 986   ;
                                    ; 987   ;This code handles EOF case 0.0.  No 1st wrap, no flush, low byte valid.
                                    ; 988   ;
                                    ; 989
                                    ; 990   ;Branch on first overflow check. Increment RDCUR, put result in TEMP.
U 0157, 1620,1304,7657,7777,7760,3400   ; 991   V13NWR: J/V13NWA,COND/ZERO,ALUA/RDCUR,ALUB/TEMP,ALUFN/ADDZA,ALUDEST/BRAM,CRY_1
                                    ; 992
                                    ; 993   =0
                                    ; 994   ;Compute TEMP(=RDCUR)-RWRAP for second wrap check
U 0162, 1642,1121,6377,7777,7760,3400   ; 995   V13NWA: J/V13NWB,ALUA/TEMP,ALUB/RWRAP,ALUFN/SUBAB,ALUDEST/NOP,CRY_1
                                    ; 996
                                    ; 997   ;We hit the deposit limit.  Jump to abort code after preparing RDCUR wrap chk
U 0163, 2352,0121,4317,7777,7760,3400   ; 998           J/V13ABT,ALUA/RWRAP,ALUB/RDBASE,ALUFN/SUBAB,ALUDEST/NOP
                                    ; 999   =
                                    ; 1000  ;Branch on result of second wrap check. Also start check for Q-reg negative
U 0164, 1660,0132,7777,7777,7760,3400   ; 1001  V13NWB: J/V13NWC,COND/ZERO,ALUA/TEMP,ALUB/TEMP,ALUFN/Q,ALUDEST/NOP
                                    ; 1002
                                    ; 1003  =0
                                    ; 1004
                                    ; 1005  ;We do not need to wrap. If we don't overflow, we know that RDCUR will be
                                    ; 1006  ; TEMP and that RDBASE will be the (present) value of RDCUR.  We prepare for
                                    ; 1007  ; the second overflow check by computing RDLIMIT-TEMP. We then jump to the
                                    ; 1008  ; V13N2A section of code to finish up.  We are futher branching on whether
                                    ; 1009  ; the Q-reg (length) is negative.  If so, we need to add the length of the RX
                                    ; 1010  ; buffer to Q to get the correct length.
                                    ; 1011
U 0166, 2000,3121,7737,7777,7760,3400   ; 1012  V13NWC: J/V13N2A,COND/NEG,ALUA/RDLIMIT,ALUB/TEMP,ALUFN/SUBAB,ALUDEST/NOP,CRY_1
                                    ; 1013
                                    ; 1014  ;We need to wrap. If we don't overflow, we know that RDCUR will be TWRAP
                                    ; 1015  ; and that RDBASE will be the old value of RDCUR.  At this point we prepare
                                    ; 1016  ; for the second overflow check by computing RDLIMIT-TWRAP. We then jump to
                                    ; 1017  ; the V13W2A section of code to finish up. We are futher branching on whether
                                    ; 1018  ; the Q-reg (length) is negative.  If so, we need to add the length of the RX
                                    ; 1019  ; buffer to Q to get the correct length.
                                    ; 1020
U 0167, 1700,3121,2737,7777,7760,3400   ; 1021          J/V13W2A,COND/NEG,ALUA/RDLIMIT,ALUB/TWRAP,ALUFN/SUBAB,ALUDEST/NOP,CRY_1
                                    ; 1022  =
                                    ; 1023
```

```
                                    ; 1024
                                    ; 1025  ;Vector Request 13 - EOF, low byte valid.
                                    ; 1026  ;---------------------------------------
                                    ; 1027
                                    ; 1028  ;Here to finish up EOF case 0.0 if we had to wrap on the second increment.
                                    ; 1029
                                    ; 1030  =0
                                    ; 1031  ;Test for overflow, put RDBASE onto data bus. Q contains correct length.
U 0170, 1740,0134,7607,7777,7760,3400   ; 1032  V13W2A: J/V13W2C,COND/ZERO,DALU_READ,ALUA/RDBASE,ALUB/TEMP,ALUFN/A,ALUDEST/NOP
                                    ; 1033
                                    ; 1034  ;Test for overflow, put RDBASE onto data bus. Q needs correcting (is negative)
U 0171, 1720,0134,7607,7777,7760,3400   ; 1035          J/V13W2B,COND/ZERO,DALU_READ,ALUA/RDBASE,ALUB/TEMP,ALUFN/A,ALUDEST/NOP
                                    ; 1036  =
                                    ; 1037  =0
                                    ; 1038  ;Put Q+<length of RX> on data bus and write length into buffer
                                    ; 1039  V13W2B: J/V13W2D,DALU_READ,BUFR_LOAD,ALUA/TWRAP,ALUB/TEMP,ALUFN/ADDAQ,
U 0172, 1652,0100,7526,7777,7760,3400   ; 1040           ALUDEST/NOP
                                    ; 1041
                                    ; 1042  ;We hit the deposit limit.  Jump to abort code after preparing RDCUR wrap chk
U 0173, 2352,0121,4317,7777,7760,3400   ; 1043          J/V13ABT,ALUA/RWRAP,ALUB/RDBASE,ALUFN/SUBAB,ALUDEST/NOP
                                    ; 1044  =
                                    ; 1045
                                    ; 1046  =0
                                    ; 1047  ;Put Q on data bus and write length into buffer
U 0174, 1652,0132,7766,7777,7760,3400   ; 1048  V13W2C: J/V13W2D,DALU_READ,BUFR_LOAD,ALUA/TEMP,ALUB/TEMP,ALUFN/Q,ALUDEST/NOP
                                    ; 1049
                                    ; 1050  ;We hit the deposit limit.  Jump to abort code after preparing RDCUR wrap chk
U 0175, 2352,0121,4317,7777,7760,3400   ; 1051          J/V13ABT,ALUA/RWRAP,ALUB/RDBASE,ALUFN/SUBAB,ALUDEST/NOP
                                    ; 1052  =
                                    ; 1053
                                    ; 1054  ;Do RDBASE := RDCUR.  Say a packet is available.
U 0165, 1762,0334,4257,7777,7370,3400   ; 1055  V13W2D: SET_STATUS,RX_PKT_AVAIL,ALUA/RDCUR,ALUB/RDBASE,ALUFN/A,ALUDEST/BRAM
                                    ; 1056
                                    ; 1057  ;Do RDCUR := TWRAP and vector
U 0176, 3406,0334,5137,7777,7760,3400   ; 1058          JUMP_VECTOR,ALUA/TWRAP,ALUB/RDCUR,ALUFN/A,ALUDEST/BRAM
                                    ; 1059
```

```
                                    ; 1060
                                    ; 1061  ;Vector Request 13 - EOF, low byte valid.
                                    ; 1062  ;---------------------------------------
                                    ; 1063  ;
                                    ; 1064  ;Here to finish up EOF case 0.0 if we did not wrap on the second increment.
                                    ; 1065
                                    ; 1066  =0
                                    ; 1067  ;Test for overflow, put RDBASE onto data bus.  Q is okay
U 0200, 2040,0134,7607,7777,7760,3400  ; 1068  V13N2A: J/V13N2C,COND/ZERO,DALU_READ,ALUA/RDBASE,ALUB/TEMP,ALUFN/A,ALUDEST/NOP
                                    ; 1069
                                    ; 1070  ;Test for overflow, put RDBASE onto data bus. Q is negative and needs fixing.
U 0201, 2020,0134,7607,7777,7760,3400  ; 1071          J/V13N2B,COND/ZERO,DALU_READ,ALUA/RDBASE,ALUB/TEMP,ALUFN/A,ALUDEST/NOP
                                    ; 1072  =
                                    ; 1073  =0
                                    ; 1074  ;Put Q+<RX length> on data bus and write length into buffer
                                    ; 1075  V13N2B: J/V13N2D,DALU_READ,BUFR_LOAD,ALUA/TWRAP,ALUB/TEMP,ALUFN/ADDAQ,
U 0202, 1772,0100,7526,7777,7760,3400  ; 1076           ALUDEST/NOP
                                    ; 1077
                                    ; 1078  ;We hit the deposit limit.  Jump to abort code after preparing RDCUR wrap chk
U 0203, 2352,0121,4317,7777,7760,3400  ; 1079          J/V13ABT,ALUA/RWRAP,ALUB/RDBASE,ALUFN/SUBAB,ALUDEST/NOP
                                    ; 1080  =
                                    ; 1081  =0
                                    ; 1082  ;Put Q on data bus and write length into buffer
U 0204, 1772,0132,7766,7777,7760,3400  ; 1083  V13N2C: J/V13N2D,DALU_READ,BUFR_LOAD,ALUA/TEMP,ALUB/TEMP,ALUFN/Q,ALUDEST/NOP
                                    ; 1084
                                    ; 1085  ;We hit the deposit limit.  Jump to abort code after preparing RDCUR wrap chk
U 0205, 2352,0121,4317,7777,7760,3400  ; 1086          J/V13ABT,ALUA/RWRAP,ALUB/RDBASE,ALUFN/SUBAB,ALUDEST/NOP
                                    ; 1087  =
                                    ; 1088
                                    ; 1089  ;Do RDBASE := RDCUR, say that a packet is available
U 0177, 2062,0334,4257,7777,7370,3400  ; 1090  V13N2D: SET_STATUS,RX_PKT_AVAIL,ALUA/RDCUR,ALUB/RDBASE,ALUFN/A,ALUDEST/BRAM
                                    ; 1091
                                    ; 1092  ;Do RDCUR := TEMP and vector
U 0206, 3406,0334,5377,7777,7760,3400  ; 1093          JUMP_VECTOR,ALUA/TEMP,ALUB/RDCUR,ALUFN/A,ALUDEST/BRAM
                                    ; 1094
```

```
                                    ; 1095
                                    ; 1096  ;Vector Request 13 - EOF, low byte NOT valid
                                    ; 1097  ;-----------------------------------------
                                    ; 1098  ;
                                    ; 1099  ;This code handles EOF case 0.0.  No 1st wrap, no flush, low byte NOT valid.
                                    ; 1100  ;The only difference between this page and the one starting with V13NWR
                                    ; 1101  ; is that V13NWB puts ~Q on the bus and into the Q-reg.
                                    ; 1102
                                    ; 1103  ;Branch on first overflow check. Increment RDCUR, put result in TEMP.
U 0207, 2100,1304,7657,7777,7760,3400  ; 1104  V3BNWR: J/V3BNWA,COND/ZERO,ALUA/RDCUR,ALUB/TEMP,ALUFN/ADDZA,ALUDEST/BRAM,CRY_1
                                    ; 1105
                                    ; 1106  =0
                                    ; 1107  ;Compute TEMP(=RDCUR)-RWRAP for second wrap check
U 0210, 2122,1121,6377,7777,7760,3400  ; 1108  V3BNWA: J/V3BNWB,ALUA/TEMP,ALUB/RWRAP,ALUFN/SUBAB,ALUDEST/NOP,CRY_1
                                    ; 1109
                                    ; 1110  ;We hit the deposit limit.  Jump to abort code after preparing RDCUR wrap chk
U 0211, 2352,0121,4317,7777,7760,3400  ; 1111          J/V13ABT,ALUA/RWRAP,ALUB/RDBASE,ALUFN/SUBAB,ALUDEST/NOP
                                    ; 1112  =
                                    ; 1113  ;Branch on result of second wrap check.  If Q as it stands in Q-reg is
                                    ; 1114  ; negative, we will need to add it to the length of the RX buffer to get the
                                    ; 1115  ; correct length.  Since we are handling low byte invalid, we want the ultimate
                                    ; 1116  ; length to be negative.  To that end, negate Q for our test and write the
                                    ; 1117  ; result back to the Q-reg.
U 0212, 2140,1022,7777,7777,7760,3400  ; 1118  V3BNWB: J/V3BNWC,COND/ZERO,ALUA/TEMP,ALUB/TEMP,ALUFN/SUBZQ,ALUDEST/Q,CRY_1
                                    ; 1119
                                    ; 1120  =0
                                    ; 1121
                                    ; 1122  ;We do not need to wrap. If we don't overflow, we know that RDCUR will be
                                    ; 1123  ; TEMP and that RDBASE will be the (present) value of RDCUR.  We prepare for
                                    ; 1124  ; the second overflow check by computing RDLIMIT-TEMP. We then jump to the
                                    ; 1125  ; V3BN2A section of code to finish up.  We are futher branching on whether
                                    ; 1126  ; the Q-reg (length) was originally negative.  If so, we need to add the
                                    ; 1127  ; length of the RX buffer to Q to get the correct length.
                                    ; 1128
U 0214, 2260,3121,7737,7777,7760,3400  ; 1129  V3BNWC: J/V3BN2A,COND/NEG,ALUA/RDLIMIT,ALUB/TEMP,ALUFN/SUBAB,ALUDEST/NOP,CRY_1
                                    ; 1130
                                    ; 1131  ;We need to wrap. If we don't overflow, we know that RDCUR will be TWRAP
                                    ; 1132  ; and that RDBASE will be the old value of RDCUR.  At this point we prepare
                                    ; 1133  ; for the second overflow check by computing RDLIMIT-TWRAP. We then jump to
                                    ; 1134  ; the V3BW2A section of code to finish up. We are futher branching on whether
                                    ; 1135  ; the Q-reg (length) was originally negative.  If so, we need to add the
                                    ; 1136  ; length of the RX buffer to Q to get the correct length.
                                    ; 1137
U 0215, 2160,3121,2737,7777,7760,3400  ; 1138          J/V3BW2A,COND/NEG,ALUA/RDLIMIT,ALUB/TWRAP,ALUFN/SUBAB,ALUDEST/NOP,CRY_1
                                    ; 1139  =
                                    ; 1140
```

```
                                    ; 1141
                                    ; 1142  ;Vector Request 13 - EOF, low byte NOT valid
                                    ; 1143  ;-------------------------------------------
                                    ; 1144
                                    ; 1145  ;Here to finish up EOF case 0.0 if we had to wrap on the second increment.
                                    ; 1146  ;This is the low byte invalid code. It differs from the page starting with
                                    ; 1147  ; V13W2A in that the two instructions at V13W2A are swapped around and in
                                    ; 1148  ; that the instruction at V13W2B was also changed.  Everything else is the
                                    ; 1149  ; same.
                                    ; 1150
                                    ; 1151  =0
                                    ; 1152  V3BW2A:
                                    ; 1153  ;Test for overflow, put RDBASE onto data bus. Q needs correcting (was negative)
U 0216, 2200,0134,7607,7777,7760,3400  ; 1154          J/V3BW2B,COND/ZERO,DALU_READ,ALUA/RDBASE,ALUB/TEMP,ALUFN/A,ALUDEST/NOP
                                    ; 1155
                                    ; 1156  ;Test for overflow, put RDBASE onto data bus. Q contains correct length.
U 0217, 2220,0134,7607,7777,7760,3400  ; 1157          J/V3BW2C,COND/ZERO,DALU_READ,ALUA/RDBASE,ALUB/TEMP,ALUFN/A,ALUDEST/NOP
                                    ; 1158  =
                                    ; 1159  =0
                                    ; 1160  ;Put Q-<length of RX> on data bus and write a negated length into buffer
                                    ; 1161  V3BW2B: J/V3BW2D,DALU_READ,BUFR_LOAD,ALUA/TWRAP,ALUB/TEMP,ALUFN/SUBQA,
U 0220, 2132,1110,7526,7777,7760,3400  ; 1162           ALUDEST/NOP,CRY_1
                                    ; 1163
                                    ; 1164  ;We hit the deposit limit.  Jump to abort code after preparing RDCUR wrap chk
U 0221, 2352,0121,4317,7777,7760,3400  ; 1165          J/V13ABT,ALUA/RWRAP,ALUB/RDBASE,ALUFN/SUBAB,ALUDEST/NOP
                                    ; 1166  =
                                    ; 1167
                                    ; 1168  =0
                                    ; 1169  ;Put Q on data bus and write a negated length into buffer
U 0222, 2132,0132,7766,7777,7760,3400  ; 1170  V3BW2C: J/V3BW2D,DALU_READ,BUFR_LOAD,ALUA/TEMP,ALUB/TEMP,ALUFN/Q,ALUDEST/NOP
                                    ; 1171
                                    ; 1172  ;We hit the deposit limit.  Jump to abort code after preparing RDCUR wrap chk
U 0223, 2352,0121,4317,7777,7760,3400  ; 1173          J/V13ABT,ALUA/RWRAP,ALUB/RDBASE,ALUFN/SUBAB,ALUDEST/NOP
                                    ; 1174  =
                                    ; 1175
                                    ; 1176  ;Do RDBASE := RDCUR, say a packet is available
U 0213, 2242,0334,4257,7777,7370,3400  ; 1177  V3BW2D: SET_STATUS,RX_PKT_AVAIL,ALUA/RDCUR,ALUB/RDBASE,ALUFN/A,ALUDEST/BRAM
                                    ; 1178
                                    ; 1179  ;Do RDCUR := TWRAP and vector
U 0224, 3406,0334,5137,7777,7760,3400  ; 1180          JUMP_VECTOR,ALUA/TWRAP,ALUB/RDCUR,ALUFN/A,ALUDEST/BRAM
                                    ; 1181
```

```
                                    ; 1182
                                    ; 1183  ;Vector Request 13 - EOF, low byte NOT valid
                                    ; 1184  ;-----------------------------------------
                                    ; 1185  ;
                                    ; 1186  ;Here to finish up EOF case 0.0 if we did not wrap on the second increment.
                                    ; 1187  ;This is the low byte invalid code. It differs from the page starting with
                                    ; 1188  ; V13N2A in that the two instructions at V13N2A are swapped around and in
                                    ; 1189  ; that the instruction at V13N2B was also changed.  Everything else is the
                                    ; 1190  ; same.
                                    ; 1191
                                    ; 1192  =0
                                    ; 1193  ;Test for overflow, put RDBASE onto data bus. Q is negative and needs fixing.
U 0226, 2300,0134,7607,7777,7760,3400  ; 1194  V3BN2A: J/V3BN2B,COND/ZERO,DALU_READ,ALUA/RDBASE,ALUB/TEMP,ALUFN/A,ALUDEST/NOP
                                    ; 1195
                                    ; 1196  ;Test for overflow, put RDBASE onto data bus.  Q is okay
U 0227, 2320,0134,7607,7777,7760,3400  ; 1197          J/V3BN2C,COND/ZERO,DALU_READ,ALUA/RDBASE,ALUB/TEMP,ALUFN/A,ALUDEST/NOP
                                    ; 1198  =
                                    ; 1199
                                    ; 1200  =0
                                    ; 1201  ;Put Q-<RX length> on data bus and write a negated length into buffer
                                    ; 1202  V3BN2B: J/V3BN2D,DALU_READ,BUFR_LOAD,ALUA/TWRAP,ALUB/TEMP,ALUFN/SUBQA,
U 0230, 2252,1110,7526,7777,7760,3400  ; 1203           ALUDEST/NOP,CRY_1
                                    ; 1204
                                    ; 1205  ;We hit the deposit limit.  Jump to abort code after preparing RDCUR wrap chk
U 0231, 2352,0121,4317,7777,7760,3400  ; 1206          J/V13ABT,ALUA/RWRAP,ALUB/RDBASE,ALUFN/SUBAB,ALUDEST/NOP
                                    ; 1207  =
                                    ; 1208  =0
                                    ; 1209  ;Put Q on data bus and write a negated length into buffer
U 0232, 2252,0132,7766,7777,7760,3400  ; 1210  V3BN2C: J/V3BN2D,DALU_READ,BUFR_LOAD,ALUA/TEMP,ALUB/TEMP,ALUFN/Q,ALUDEST/NOP
                                    ; 1211
                                    ; 1212  ;We hit the deposit limit.  Jump to abort code after preparing RDCUR wrap chk
U 0233, 2352,0121,4317,7777,7760,3400  ; 1213          J/V13ABT,ALUA/RWRAP,ALUB/RDBASE,ALUFN/SUBAB,ALUDEST/NOP
                                    ; 1214  =
                                    ; 1215
                                    ; 1216  ;Do RDBASE := RDCUR, say a packet is available
U 0225, 2342,0334,4257,7777,7370,3400  ; 1217  V3BN2D: SET_STATUS,RX_PKT_AVAIL,ALUA/RDCUR,ALUB/RDBASE,ALUFN/A,ALUDEST/BRAM
                                    ; 1218
                                    ; 1219  ;Do RDCUR := TEMP and vector
U 0234, 3406,0334,5377,7777,7760,3400  ; 1220          JUMP_VECTOR,ALUA/TEMP,ALUB/RDCUR,ALUFN/A,ALUDEST/BRAM
                                    ; 1221
                                    ; 1222
```

```
                                    ; 1223
                                    ; 1224  ;Vector Request 13 - Abort a packet.
                                    ; 1225  ;----------------------------------
                                    ; 1226  ;
                                    ; 1227  ;All RX abort code comes here after doing (RWRAP - (RDBASE+1)) for RDCUR wrap
                                    ; 1228  ; check.
                                    ; 1229  ;
                                    ; 1230  ;Branch on wrap, clear bad address line
U 0235, 2360,0147,7777,7577,7760,3400  ; 1231  V13ABT: J/V13ABA,COND/ZERO,NI_BADRS_RST
                                    ; 1232
                                    ; 1233  =0
                                    ; 1234  ;No need to wrap, do RDCUR := RDBASE + 1.
U 0236, 3406,1304,5217,7777,7760,3400  ; 1235  V13ABA: JUMP_VECTOR,ALUA/RDBASE,ALUB/RDCUR,ALUFN/ADD2A,ALUDEST/BRAM,CRY_1
                                    ; 1236
                                    ; 1237  ;Here to set RDCUR to TWRAP and vector
U 0237, 3406,0334,5137,7777,7760,3400  ; 1238          JUMP_VECTOR,ALUA/TWRAP,ALUB/RDCUR,ALUFN/A,ALUDEST/BRAM
                                    ; 1239  =
                                    ; 1240
```

```
                                        ; 1241
                                        ; 1242  ;Vector Request 15 - TX abort
                                        ; 1243  ;----------------------------
                                        ; 1244  ; 4 ins
                                        ; 1245
                                        ; 1246  ;A transfer was aborted, presumably because of a collision.  We turn off
                                        ; 1247  ; TX_REQ_ENABLE and reset TWCUR.  The IDLE routine will restart the NI.
                                        ; 1248
                                        ; 1249  ;Shut off NI and do TWCUR := TWBASE.
                                        ; 1250  ;V..15: J/V15,CLEAR_STATUS,TX_REQ_ENABLE,ALUA/TWBASE,ALUB/TWCUR,ALUFN/A,
                                        ; 1251  ;        ALUDEST/BRAM
                                        ; 1252
                                        ; 1253  ;Waste a cycle to make sure TX_REQ_ENABLE goes down
U 0240, 3406,0147,7777,7777,7760,3400   ; 1254  V15:    JUMP_VECTOR
                                        ; 1255
```

```
                                    ; 1256
                                    ; 1257  ;Vector Request 16 - Mport Load (RX buffer to Barrel Board)
                                    ; 1258  ;-------------------------------------------------------
                                    ; 1259  ; 5 ins. normal, 3 ins. at finish
                                    ; 1260
                                    ; 1261  ;Do RWCUR-RWTOP to see if we already sent last data word
                                    ; 1262  ;V..16: J/V16,ALUA/RWCUR,ALUB/RWTOP,ALUFN/SUBAB,ALUDEST/NOP,CRY_1
                                    ; 1263
                                    ; 1264  ;Branch on transfer done.  Put RWCUR on data bus and increment RWCUR
                                    ; 1265  V16:    J/V16A,COND/ZERO,DALU_READ,ALUA/RWCUR,ALUB/RWCUR,ALUFN/ADDZA,
U 0241, 2420,1204,4627,7777,7760,3400  ; 1266           ALUDEST/BRAMA,CRY_1
                                    ; 1267
                                    ; 1268  =0
                                    ; 1269  ;Send data. Do RWCUR-RWRAP for wrap check
                                    ; 1270  V16A:   J/V16B,MPORT_LOAD,BUFR_READ,ALUA/RWCUR,ALUB/RWRAP,ALUFN/SUBAB,
U 0242, 2442,1121,6235,7377,7760,3400  ; 1271           ALUDEST/NOP,CRY_1
                                    ; 1272
                                    ; 1273  ;Done. Must clear XIPR, decrement RWCUR, and vector
                                    ; 1274          JUMP_VECTOR,CLEAR_XIPR,ALUA/RWCUR,ALUB/RWCUR,ALUFN/SUBZA,
U 0243, 3406,0314,4637,7777,7760,7400  ; 1275           ALUDEST/BRAM,CRY_0
                                    ; 1276  =
                                    ; 1277  ;Branch on result of wrap check
U 0244, 2460,0147,7777,7777,7760,3400  ; 1278  V16B:   J/V16C,COND/ZERO
                                    ; 1279
                                    ; 1280  =0
                                    ; 1281  ;No wrap needed, just vector
U 0246, 3406,0147,7777,7777,7760,3400  ; 1282  V16C:   JUMP_VECTOR
                                    ; 1283
                                    ; 1284  ;Here to wrap RWCUR to TWRAP and vector
U 0247, 3406,0334,4537,7777,7760,3400  ; 1285          JUMP_VECTOR,ALUA/TWRAP,ALUB/RWCUR,ALUFN/A,ALUDEST/BRAM
                                    ; 1286  =
                                    ; 1287
```

```
                                        ; 1288
                                        ; 1289  ;Vector Request 17 - Mport Read (Barrel Board to transmit buffer)
                                        ; 1290  ;-------------------------------------------------------------
                                        ; 1291  ; 4 ins
                                        ; 1292
                                        ; 1293  ;We don't need to pay attention to low byte valid in this routine since
                                        ; 1294  ; the massbus write termination routine in RCON handles that condition.
                                        ; 1295
                                        ; 1296  ; Increment TDCUR and load address onto data bus
                                        ; 1297  ;V..17: J/V17,DALU_READ,ALUA/TDCUR,ALUB/TDCUR,ALUFN/ADDZB,ALUDEST/BRAMA,CRY_1
                                        ; 1298
                                        ; 1299  ; Assert buffer load, mport read.  Compute TWRAP-TDCUR to check for wrap
                                        ; 1300  V17:    MPORT_READ,BUFR_LOAD,ALUA/TWRAP,ALUB/TDCUR,ALUFN/SUBAB,
U 0245, 2502,1121,1536,6777,7760,3400   ; 1301           ALUDEST/NOP,CRY_1
                                        ; 1302
                                        ; 1303  ;Branch on previous result. If ZERO, then we need to wrap.
U 0250, 2520,0147,7777,7777,7760,3400   ; 1304          J/V17A,COND/ZERO
                                        ; 1305  =0
                                        ; 1306  ;Don't need to wrap, return to jump vector table
U 0252, 3406,0147,7777,7777,7760,3400   ; 1307  V17A:   JUMP_VECTOR
                                        ; 1308
                                        ; 1309  ;Wrap deposit pointer (set to zero) and return to jump vector table.
U 0253, 3406,0347,1777,7777,7760,3400   ; 1310          JUMP_VECTOR,ALUA/TEMP,ALUB/TDCUR,ALUFN/Z,ALUDEST/BRAM
                                        ; 1311  =
                                        ; 1312
```

```
                                        ; 1313  .TOC "Register Control Board Requests"
                                        ; 1314
                                        ; 1315  ;RCON Request Dispatch Table
                                        ; 1316  ;---------------------------
                                        ; 1317
                                        ; 1318  =11110000
                                        ; 1319  R.FIELD:
                                        ; 1320
                                        ; 1321  ;Requests 0-7 require an XPORT_READ to complete
                                        ; 1322
                                        ; 1323  ;00 - Ethernet mask data/address load
                                        ; 1324  ;Read address or data from Xport into TEMP.
U 0360, 2512,0337,7777,3777,7760,3400   ; 1325          J/R00,XPORT_READ,ALUA/TEMP,ALUB/TEMP,ALUFN/D,ALUDEST/BRAM
                                        ; 1326
                                        ; 1327  ;01 - Buffer data load
                                        ; 1328  ;Put buffer address in SNIFF on data bus
U 0361, 2572,0134,7667,7777,7760,3400   ; 1329          J/R01,DALU_READ,ALUA/SNIFF,ALUB/TEMP,ALUFN/A,ALUDEST/NOP
                                        ; 1330
                                        ; 1331  ;02 - Buffer address load
                                        ; 1332  ;Read address from Xport into SNIFF.
U 0362, 2622,0337,5777,3777,7760,3400   ; 1333          J/R02,XPORT_READ,ALUA/TEMP,ALUB/SNIFF,ALUFN/D,ALUDEST/BRAM
                                        ; 1334
                                        ; 1335  ;03 - Drive clear.
U 0363, 0002,0147,7777,3777,7760,3400   ; 1336          J/INIT,XPORT_READ
                                        ; 1337
                                        ; 1338  ;04 - Flush RX packet request
                                        ; 1339  ;Do RWBASE-RDBASE for empty buffer check
U 0364, 2722,1121,4177,7777,7760,3400   ; 1340          J/R04,ALUA/RWBASE,ALUB/RDBASE,ALUFN/SUBAB,ALUDEST/NOP,CRY_1
                                        ; 1341
                                        ; 1342  ;05 - 2901 Register Dump (formerly TX buffer flush)
                                        ; 1343  ;Set our counter (TEMP) to zero and do obligatory XPORT_READ
U 0365, 3212,0347,7777,3777,7760,3400   ; 1344          J/R05,XPORT_READ,ALUA/TEMP,ALUB/TEMP,ALUFN/Z,ALUDEST/BRAM
                                        ; 1345
                                        ; 1346  ;06 - Ethernet mask data read
                                        ; 1347  ;Select a cell in the Address RAM, copy that address into ADDRM
                                        ; 1348          J/R06,NI_MASK_ADRS_LOAD,XPORT_READ,ALUA/ADDRM,ALUB/ADDRM,ALUFN/D,
U 0366, 4252,0337,5677,3677,7760,3400   ; 1349           ALUDEST/BRAM
                                        ; 1350
                                        ; 1351  ;07 - Trailer load
                                        ; 1352  ;Put TDCUR on data bus and increment it
U 0367, 4262,1204,1467,7777,7760,3400   ; 1353          J/R07,DALU_READ,ALUA/TDCUR,ALUB/TDCUR,ALUFN/ADDZA,ALUDEST/BRAMA,CRY_1
                                        ; 1354
```

```
                                    ; 1355
                                    ; 1356  ;RCON Request Dispatch Table (cont'd)
                                    ; 1357  ;-----------------------------------
                                    ; 1358
                                    ; 1359  ;Requests 10-17 require XPORT_LOAD to complete
                                    ; 1360
                                    ; 1361  ;10 - Error
U 0370, 3406,0147,7777,5777,7760,3400   ; 1362          JUMP_VECTOR,XPORT_LOAD
                                    ; 1363
                                    ; 1364  ;11 - Error
U 0371, 3406,0147,7777,5777,7760,3400   ; 1365          JUMP_VECTOR,XPORT_LOAD
                                    ; 1366
                                    ; 1367  ;12 - End massbus read
                                    ; 1368  ;Put RWBASE on DBus.
U 0372, 4372,0134,7567,7777,7760,3400   ; 1369          J/R12,DALU_READ,ALUA/RWBASE,ALUB/TEMP,ALUFN/A,ALUDEST/NOP
                                    ; 1370
                                    ; 1371  ;13 - Buffer pointer read
                                    ; 1372          JUMP_VECTOR,DALU_READ,XPORT_LOAD,ALUA/SNIFF,ALUB/TEMP,ALUFN/A,
U 0373, 3406,0134,7667,5777,7760,3400   ; 1373           ALUDEST/NOP
                                    ; 1374
                                    ; 1375  ;14 - End massbus write
                                    ; 1376  ;Compute TDCUR-TDBASE-1 (uncorrected pkt length) and put in Q register.
                                    ; 1377  ; Do XPORT_LOAD to satisfy micro-machine.
U 0374, 4412,0021,0477,5777,7760,3400   ; 1378          J/R14,XPORT_LOAD,ALUA/TDCUR,ALUB/TDBASE,ALUFN/SUBAB,ALUDEST/Q,CRY_0
                                    ; 1379
                                    ; 1380  ;15 - Start massbus read
                                    ; 1381  ;Clear XIPR and put RWBASE on DBus.  Do obligatory XPORT_LOAD.
                                    ; 1382          J/R15,DALU_READ,XPORT_LOAD,CLEAR_XIPR,ALUA/RWBASE,ALUB/TEMP,ALUFN/A,
U 0375, 4602,0134,7567,5777,7760,7400   ; 1383           ALUDEST/NOP
                                    ; 1384
                                    ; 1385  ;16 - Slave ID read
U 0376, 3406,0147,7773,5777,7760,3400   ; 1386          JUMP_VECTOR,SLIDE_READ,XPORT_LOAD
                                    ; 1387
                                    ; 1388  ;17 - Packet status read
                                    ; 1389  ;Put RWBASE on data bus
U 0377, 4702,0134,7567,7777,7760,3400   ; 1390          J/R17,DALU_READ,ALUA/RWBASE,ALUB/TEMP,ALUFN/A,ALUDEST/NOP
                                    ; 1391  =
                                    ; 1392
```

```
                                    ; 1393
                                    ; 1394  ;RCON Request 00 - Ethernet Address Load
                                    ; 1395  ;-------------------------------------
                                    ; 1396  ; 5 ins
                                    ; 1397
                                    ; 1398  ;Read data from Xport into TEMP.
                                    ; 1399  ;R..00: J/R00,XPORT_READ,ALUA/TEMP,ALUB/TEMP,ALUFN/D,ALUDEST/BRAM
                                    ; 1400
                                    ; 1401  ;Use ADDRM to select address mask.
U 0251, 2542,0134,5667,7677,7760,3400  ; 1402  R00:    NI_MASK_ADRS_LOAD,DALU_READ,ALUA/ADDRM,ALUB/ADDRM,ALUFN/A,ALUDEST/NOP
                                    ; 1403
                                    ; 1404  ;Load data into the address mask.
U 0254, 2552,0134,7767,7757,7760,3400  ; 1405          NI_MASK_DATA_LOAD,DALU_READ,ALUA/TEMP,ALUB/TEMP,ALUFN/A,ALUDEST/NOP
                                    ; 1406
                                    ; 1407  ;Re-select address mask
U 0255, 2562,0134,5667,7677,7760,3400  ; 1408          NI_MASK_ADRS_LOAD,DALU_READ,ALUA/ADDRM,ALUB/ADDRM,ALUFN/A,ALUDEST/NOP
                                    ; 1409
                                    ; 1410  ;Read data from address mask, then vector
U 0256, 3406,0147,7777,5737,7760,3400  ; 1411          JUMP_VECTOR,NI_MASK_DATA_READ,XPORT_LOAD
                                    ; 1412
```

```
                                    ; 1413
                                    ; 1414  ;RCON Request 01 - Buffer Data Load
                                    ; 1415  ;---------------------------------
                                    ; 1416  ; 4 ins
                                    ; 1417
                                    ; 1418  ;We write data from the Xport into the buffer and read it back again.
                                    ; 1419  ; We assume that SNIFF has been set up previously.  Since SNIFF is the
                                    ; 1420  ; same as RWTOP, no massbus read must be in progress!
                                    ; 1421
                                    ; 1422  ;Put buffer address in SNIFF on data bus
                                    ; 1423  ;R..01: J/R01,DALU_READ,ALUA/SNIFF,ALUB/TEMP,ALUFN/A,ALUDEST/NOP
                                    ; 1424
                                    ; 1425  ;Write from Xport into buffer
U 0257, 2602,0147,7776,3777,7760,3400  ; 1426  R01:    XPORT_READ,BUFR_LOAD
                                    ; 1427
                                    ; 1428  ;Put buffer address on data bus again
U 0260, 2612,0134,7667,7777,7760,3400  ; 1429          DALU_READ,ALUA/SNIFF,ALUB/TEMP,ALUFN/A,ALUDEST/NOP
                                    ; 1430
                                    ; 1431  ;Read from buffer to Xport, then vector
U 0261, 3406,0147,7775,5777,7760,3400  ; 1432          JUMP_VECTOR,BUFR_READ,XPORT_LOAD
                                    ; 1433
```

```
                                    ; 1434
                                    ; 1435  ;RCON Request 02 - Buffer Address Load
                                    ; 1436  ;-------------------------------------
                                    ; 1437  ; Abs is 3 ins, Rel is 7 or 8
                                    ; 1438
                                    ; 1439  ;Load an address from Xport into SNIFF, read back ram buffer data.
                                    ; 1440  ; If the address is negative, then it is absolute, otherwise it is relative.
                                    ; 1441  ; No range checking is done for either type of pointer.
                                    ; 1442
                                    ; 1443  ;Read address from Xport into SNIFF.
                                    ; 1444  ;R..02: J/R02,XPORT_READ,ALUA/TEMP,ALUB/SNIFF,ALUFN/D,ALUDEST/BRAM
                                    ; 1445
                                    ; 1446  ;Branch on SNIFF being negative. Do some speed-up by putting SNIFF on DBus
U 0262, 2640,2134,7667,7777,7760,3400  ; 1447  R02:    J/R02A,COND/NEG,DALU_READ,ALUA/SNIFF,ALUB/TEMP,ALUFN/A,ALUDEST/NOP
                                    ; 1448
                                    ; 1449  =0
                                    ; 1450  ;Address was non-negative, i.e. relative.  Do SNIFF := RWBASE + SNIFF.
                                    ; 1451  ;Note that a relative offset of zero gives the user the length word.
                                    ; 1452  ; This is to be consistent with the way that the massbus read works.
U 0264, 2632,0301,5567,7777,7760,3400  ; 1453  R02A:   J/R02B,DEBUG,ALUA/RWBASE,ALUB/SNIFF,ALUFN/ADDAB,ALUDEST/BRAM
                                    ; 1454
                                    ; 1455  ;Address was negative, i.e. absolute. Load Xport and vector
U 0265, 3406,0147,7775,5777,7760,3400  ; 1456          JUMP_VECTOR,BUFR_READ,XPORT_LOAD
                                    ; 1457  =
                                    ; 1458  ;Do SNIFF - RWRAP to test for wrap, result to Q-reg
U 0263, 2662,1021,6267,7777,7760,3400  ; 1459  R02B:   DEBUG,ALUA/SNIFF,ALUB/RWRAP,ALUFN/SUBAB,ALUDEST/Q,CRY_1
                                    ; 1460
                                    ; 1461  ;Test for result being negative, do Q + TWRAP to Q-reg
U 0266, 2700,2000,7527,7777,7760,3400  ; 1462          J/R02C,COND/NEG,DEBUG,ALUA/TWRAP,ALUB/TEMP,ALUFN/ADDAQ,ALUDEST/Q
                                    ; 1463  =0
                                    ; 1464  ;Result was non-negative, i.e., we wrapped, copy Q to SNIFF and to data bus
U 0270, 2672,0332,5667,7777,7760,3400  ; 1465  R02C:   J/R02D,DALU_READ,ALUA/SNIFF,ALUB/SNIFF,ALUFN/Q,ALUDEST/BRAM
                                    ; 1466
                                    ; 1467  ;Result was negative, i.e. we did not wrap.  Put SNIFF on data bus
U 0271, 2672,0134,5667,7777,7760,3400  ; 1468          J/R02D,DALU_READ,ALUA/SNIFF,ALUB/SNIFF,ALUFN/A,ALUDEST/NOP
                                    ; 1469  =
                                    ; 1470  ;Write buffer data to Xport and vector
U 0267, 3406,0147,7775,5777,7760,3400  ; 1471  R02D:   JUMP_VECTOR,XPORT_LOAD,BUFR_READ
                                    ; 1472
```

```
                                    ; 1473
                                    ; 1474  ;RCON Request 04 - RX Flush Packet
                                    ; 1475  ;--------------------------------
                                    ; 1476  ; 12 or 13 ins
                                    ; 1477  ;
                                    ; 1478  ; Flush the current packet in the RX buffer. Most of this code is common to
                                    ; 1479  ; End Massbus Read as well (it enters at R04A). Since we know the registers
                                    ; 1480  ; RDLIMIT, RWBASE, and RWCUR end up in that order, we take a number of short
                                    ; 1481  ; cuts.
                                    ; 1482
                                    ; 1483  ;Do RWBASE-RDBASE for empty buffer check
                                    ; 1484  ;R..04: J/R04,ALUA/RWBASE,ALUB/RDBASE,ALUFN/SUBAB,ALUDEST/NOP,CRY_1
                                    ; 1485
                                    ; 1486  ;Branch on empty buffer check.  Put RWBASE on Dbus.
U 0272, 2740,0134,7567,7777,7760,3400  ; 1487  R04:    J/R04AA,COND/ZERO,DALU_READ,ALUA/RWBASE,ALUB/TEMP,ALUFN/A,ALUDEST/NOP
                                    ; 1488
                                    ; 1489  =0
                                    ; 1490  ;Something in buffer, read length word into Q-reg.
U 0274, 2732,0037,7775,7777,7760,3400  ; 1491  R04AA:  J/R04AB,BUFR_READ,ALUA/TEMP,ALUB/TEMP,ALUFN/D,ALUDEST/Q
                                    ; 1492
                                    ; 1493  ;Nothing in buffer, exit now before we mess up our pointers.  ACK the R-Board.
U 0275, 3406,0147,7777,3777,7760,3400  ; 1494          JUMP_VECTOR,XPORT_READ
                                    ; 1495  =
                                    ; 1496  ;Branch on previous result negative.  Do RDLIMIT := -Q, just in case.
                                    ; 1497  ; Do obligatory XPORT_READ.
                                    ; 1498  R04AB:  J/R04A,COND/NEG,XPORT_READ,ALUA/TEMP,ALUB/RDLIMIT,ALUFN/SUBZQ,
U 0273, 2760,3322,6777,3777,7760,3400  ; 1499           ALUDEST/BRAM,CRY_1
                                    ; 1500
                                    ; 1501  =0
                                    ; 1502  ;Length is positive. Do RWBASE + BUFFER[RWBASE], put result in RDLIMIT
U 0276, 3002,0300,6577,7777,7760,3400  ; 1503  R04A:   J/R04B,ALUA/RWBASE,ALUB/RDLIMIT,ALUFN/ADDAQ,ALUDEST/BRAM
                                    ; 1504
                                    ; 1505  ;Length is negative (odd). Compute RDLIMIT as above.
U 0277, 3002,0301,6577,7777,7760,3400  ; 1506          J/R04B,ALUA/RWBASE,ALUB/RDLIMIT,ALUFN/ADDAB,ALUDEST/BRAM
                                    ; 1507  =
                                    ; 1508
                                    ; 1509  ;Do RDLIMIT-RWRAP to see if we need to wrap, result to Q
U 0300, 3012,1021,6337,7777,7760,3400  ; 1510  R04B:   ALUA/RDLIMIT,ALUB/RWRAP,ALUFN/SUBAB,ALUDEST/Q,CRY_1
                                    ; 1511
                                    ; 1512  ;Branch on wrap.  If negative, then RDLIMIT is okay as is. Do Q := TWRAP + Q.
U 0301, 3020,2000,7537,7777,7760,3400  ; 1513          J/R04C,COND/NEG,ALUA/TWRAP,ALUB/TEMP,ALUFN/ADDAQ,ALUDEST/Q
                                    ; 1514
                                    ; 1515  =0
                                    ; 1516  ;RDLIMIT wrapped.  Do RDLIMIT := Q and join the wrap code.
U 0302, 3042,0332,6777,7777,7760,3400  ; 1517  R04C:   J/R04RDW,ALUA/TEMP,ALUB/RDLIMIT,ALUFN/Q,ALUDEST/BRAM
                                    ; 1518
                                    ; 1519  ;Here if RDLIMIT is not beyond end of buffer.  Do wrap check for RWBASE before
                                    ; 1520  ; actually setting RWBASE.  Wrap check is RWRAP - (RDLIMIT + 1).
U 0303, 3122,0121,6717,7777,7760,3400  ; 1521          J/R04RDN,ALUA/RWRAP,ALUB/RDLIMIT,ALUFN/SUBAB,ALUDEST/NOP
                                    ; 1522  =
                                    ; 1523
```

```
                                    ; 1524
                                    ; 1525  ;RCON Request 04 - RX Flush Packet - RDLIMIT wrapped
                                    ; 1526  ;-------------------------------------------------
                                    ; 1527
                                    ; 1528  ;Here if RDLIMIT wrapped.  We don't have to check wrap again
                                    ; 1529
                                    ; 1530  ;Do RWBASE := RDLIMIT + 1
U 0304, 3052,1304,3737,7777,7760,3400   ; 1531  R04RDW: ALUA/RDLIMIT,ALUB/RWBASE,ALUFN/ADDZA,ALUDEST/BRAM,CRY_1
                                    ; 1532
                                    ; 1533  ;Do RWCUR := RWBASE + 1
U 0305, 3062,1304,4577,7777,7760,3400   ; 1534          ALUA/RWBASE,ALUB/RWCUR,ALUFN/ADDZA,ALUDEST/BRAM,CRY_1
                                    ; 1535
                                    ; 1536  ;Do RWBASE-RDBASE to check for more packets
U 0306, 3072,1121,4177,7777,7760,3400   ; 1537          ALUA/RWBASE,ALUB/RDBASE,ALUFN/SUBAB,ALUDEST/NOP,CRY_1
                                    ; 1538
                                    ; 1539  ;Branch on empty check
U 0307, 3100,0147,7777,7777,7760,3400   ; 1540          J/R04RWA,COND/ZERO
                                    ; 1541  =0
                                    ; 1542  ;Here if more packets available.  Just vector
U 0310, 3406,0147,7777,7777,7760,3400   ; 1543  R04RWA: JUMP_VECTOR
                                    ; 1544
                                    ; 1545  ;No more packets.  Clear RX_PKT_AVAIL and vector
U 0311, 3406,0147,7777,7777,7360,3400   ; 1546          JUMP_VECTOR,CLEAR_STATUS,RX_PKT_AVAIL
                                    ; 1547  =
                                    ; 1548
```

```
                                    ; 1549
                                    ; 1550  ;RCON Request 04 - RX Flush Packet - RDLIMIT did not wrap
                                    ; 1551  ;-------------------------------------------------------
                                    ; 1552
                                    ; 1553  ;Here if RDLIMIT did not wrap.
                                    ; 1554
                                    ; 1555  ;Branch on RWBASE wrap check.  Do RWBASE := RDLIMIT + 1 just in case.
                                    ; 1556  R04RDN: J/R04RNA,COND/ZERO,ALUA/RDLIMIT,ALUB/RWBASE,ALUFN/ADDZA,ALUDEST/BRAM,
U 0312, 3140,1304,3737,7777,7760,3400  ; 1557           CRY_1
                                    ; 1558
                                    ; 1559  =0
                                    ; 1560  ;RWBASE did not wrap and so is valid.  Prepare RWCUR wrap check which is
                                    ; 1561  ; RWRAP - (RWBASE + 1).
U 0314, 3202,0121,3717,7777,7760,3400  ; 1562  R04RNA: J/R04RND,ALUA/RWRAP,ALUB/RWBASE,ALUFN/SUBAB,ALUDEST/NOP
                                    ; 1563
                                    ; 1564  ;RWBASE wrapped.  Do TWRAP (=RWBASE) - RDBASE for packet avail check.
U 0315, 3132,1121,4137,7777,7760,3400  ; 1565           J/R04RNB,ALUA/TWRAP,ALUB/RDBASE,ALUFN/SUBAB,ALUDEST/NOP,CRY_1
                                    ; 1566  =
                                    ; 1567
                                    ; 1568  ;Here if RWBASE wrapped
                                    ; 1569
                                    ; 1570  ;Branch on packet avail check.  Do RWBASE := TWRAP.
U 0313, 3160,0334,3537,7777,7760,3400  ; 1571  R04RNB: J/R04RNC,COND/ZERO,ALUA/TWRAP,ALUB/RWBASE,ALUFN/A,ALUDEST/BRAM
                                    ; 1572
                                    ; 1573  =0
                                    ; 1574  ;Packets still available.  Do RWCUR := RWBASE + 1, then vector
U 0316, 3406,1304,4577,7777,7760,3400  ; 1575  R04RNC: JUMP_VECTOR,ALUA/RWBASE,ALUB/RWCUR,ALUFN/ADDZA,ALUDEST/BRAM,CRY_1
                                    ; 1576
                                    ; 1577  ;No more packets.  Clear RX_PKT_AVAIL, do RWCUR := RWBASE + 1, then vector
                                    ; 1578           JUMP_VECTOR,CLEAR_STATUS,RX_PKT_AVAIL,ALUA/RWBASE,ALUB/RWCUR,
U 0317, 3406,1304,4577,7777,7360,3400  ; 1579           ALUFN/ADDZA,ALUDEST/BRAM,CRY_1
                                    ; 1580  =
                                    ; 1581
                                    ; 1582  ;Here if RWBASE did not wrap
                                    ; 1583
                                    ; 1584  ;Branch on RWCUR wrap check.  Do RWBASE - RDBASE to pkt avail check
                                    ; 1585  R04RND: J/R04RNE,COND/ZERO,ALUA/RWBASE,ALUB/RDBASE,ALUFN/SUBAB,ALUDEST/NOP,
U 0320, 3220,1121,4177,7777,7760,3400  ; 1586           CRY_1
                                    ; 1587
                                    ; 1588  =0
                                    ; 1589  ;RWCUR did not wrap.  Do RWCUR := RWBASE + 1, branch on pkt avail check
                                    ; 1590  R04RNE: J/R04RNF,COND/ZERO,ALUA/RWBASE,ALUB/RWCUR,ALUFN/ADDZA,ALUDEST/BRAM,
U 0322, 3240,1304,4577,7777,7760,3400  ; 1591           CRY_1
                                    ; 1592
                                    ; 1593  ;RWCUR wrapped.  Do RWCUR := TWRAP, branch on pkt avail check
U 0323, 3240,0334,4537,7777,7760,3400  ; 1594           J/R04RNF,COND/ZERO,ALUA/TWRAP,ALUB/RWCUR,ALUFN/A,ALUDEST/BRAM
                                    ; 1595  =
                                    ; 1596  =0
                                    ; 1597  ;More packets waiting, just vector
U 0324, 3406,0147,7777,7777,7760,3400  ; 1598  R04RNF: JUMP_VECTOR
                                    ; 1599
                                    ; 1600  ;No more packets, clear RX_PKT_AVAIL and vector
U 0325, 3406,0147,7777,7777,7360,3400  ; 1601           JUMP_VECTOR,CLEAR_STATUS,RX_PKT_AVAIL
                                    ; 1602  =
                                    ; 1603
```

```
                                    ; 1604
                                    ; 1605  ;RCON Request 05 - Dump 2901 Registers to RAM
                                    ; 1606  ;-----------------------------------------
                                    ; 1607  ; 33 instructions
                                    ; 1608
                                    ; 1609  ;We write 2901 registers 0-17 into data RAM locations 0-17.  This is meant
                                    ; 1610  ; solely for debugging the VIA board microcode.  This RCON vector used to
                                    ; 1611  ; be TX packet flush, a thoroughly useless function.
                                    ; 1612
                                    ; 1613  ;Some useful macros.  NEXT_REG selects a RAM address and DUMP_REG writes the
                                    ; 1614  ; corresponding register into that RAM location.
                                    ; 1615
                                    ; 1616  NEXT_REG        "DALU_READ,ALUA/TEMP,ALUB/TEMP,ALUFN/ADDZA,ALUDEST/BRAMA,CRY_1"
                                    ; 1617  DUMP_REG        "DALU_READ,BUFR_LOAD,ALUB/TEMP,ALUFN/A,ALUDEST/NOP"
                                    ; 1618
                                    ; 1619  ;Set our counter (TEMP) to zero and do obligatory XPORT_READ
                                    ; 1620  ;R..05: J/R05,XPORT_READ,ALUA/TEMP,ALUB/TEMP,ALUFN/Z,ALUDEST/BRAM
                                    ; 1621
U 0321, 3262,1204,7767,7777,7760,3400  ; 1622  R05:    NEXT_REG                ;Select location 0 in data RAM
U 0326, 3272,0134,7406,7777,7760,3400  ; 1623          DUMP_REG,ALUA/0         ;Write Register 0 to location 0
U 0327, 3302,1204,7767,7777,7760,3400  ; 1624          NEXT_REG                ;Etc...
U 0330, 3312,0134,7426,7777,7760,3400  ; 1625          DUMP_REG,ALUA/1
U 0331, 3322,1204,7767,7777,7760,3400  ; 1626          NEXT_REG
U 0332, 3332,0134,7446,7777,7760,3400  ; 1627          DUMP_REG,ALUA/2
U 0333, 3342,1204,7767,7777,7760,3400  ; 1628          NEXT_REG
U 0334, 3352,0134,7466,7777,7760,3400  ; 1629          DUMP_REG,ALUA/3
U 0335, 3362,1204,7767,7777,7760,3400  ; 1630          NEXT_REG
U 0336, 3372,0134,7506,7777,7760,3400  ; 1631          DUMP_REG,ALUA/4
U 0337, 4002,1204,7767,7777,7760,3400  ; 1632          NEXT_REG
U 0400, 4012,0134,7526,7777,7760,3400  ; 1633          DUMP_REG,ALUA/5
U 0401, 4022,1204,7767,7777,7760,3400  ; 1634          NEXT_REG
U 0402, 4032,0134,7546,7777,7760,3400  ; 1635          DUMP_REG,ALUA/6
U 0403, 4042,1204,7767,7777,7760,3400  ; 1636          NEXT_REG
U 0404, 4052,0134,7566,7777,7760,3400  ; 1637          DUMP_REG,ALUA/7
U 0405, 4062,1204,7767,7777,7760,3400  ; 1638          NEXT_REG
U 0406, 4072,0134,7606,7777,7760,3400  ; 1639          DUMP_REG,ALUA/10
U 0407, 4102,1204,7767,7777,7760,3400  ; 1640          NEXT_REG
U 0410, 4112,0134,7626,7777,7760,3400  ; 1641          DUMP_REG,ALUA/11
U 0411, 4122,1204,7767,7777,7760,3400  ; 1642          NEXT_REG
U 0412, 4132,0134,7646,7777,7760,3400  ; 1643          DUMP_REG,ALUA/12
U 0413, 4142,1204,7767,7777,7760,3400  ; 1644          NEXT_REG
U 0414, 4152,0134,7666,7777,7760,3400  ; 1645          DUMP_REG,ALUA/13
U 0415, 4162,1204,7767,7777,7760,3400  ; 1646          NEXT_REG
U 0416, 4172,0134,7706,7777,7760,3400  ; 1647          DUMP_REG,ALUA/14
U 0417, 4202,1204,7767,7777,7760,3400  ; 1648          NEXT_REG
U 0420, 4212,0134,7726,7777,7760,3400  ; 1649          DUMP_REG,ALUA/15
U 0421, 4222,1204,7767,7777,7760,3400  ; 1650          NEXT_REG
U 0422, 4232,0134,7746,7777,7760,3400  ; 1651          DUMP_REG,ALUA/16
U 0423, 4242,1204,7767,7777,7760,3400  ; 1652          NEXT_REG
U 0424, 3406,0134,7766,7777,7760,3400  ; 1653          JUMP_VECTOR,DUMP_REG,ALUA/17
                                    ; 1654
```

```
                                    ; 1655
                                    ; 1656  ;RCON Request 06 - Ethernet Address Read
                                    ; 1657  ;--------------------------------------
                                    ; 1658  ; 2 ins
                                    ; 1659
                                    ; 1660  ;Select a cell in the Address RAM, copy that address into ADDRM
                                    ; 1661  ;R..06: J/R06,NI_MASK_ADRS_LOAD,XPORT_READ,ALUA/ADDRM,ALUB/ADDRM,ALUFN/D,
                                    ; 1662  ;        ALUDEST/BRAM
                                    ; 1663
                                    ; 1664  ;Load Xport with address RAM data, then vector
U 0425, 3406,0147,7777,5737,7760,3400  ; 1665  R06:    JUMP_VECTOR,NI_MASK_DATA_READ,XPORT_LOAD
                                    ; 1666
```

```
                                    ; 1667
                                    ; 1668  ;RCON Request 07 - Trailer Load and End Massbus Write
                                    ; 1669  ;-------------------------------------------------
                                    ; 1670  ; 11 or 12 ins
                                    ; 1671  ;
                                    ; 1672  ;Read 16 bits of trailer data from Xport to buffer, then join code in RCON 14
                                    ; 1673  ; to finish a massbus write.  For speed we duplicate some instructions.
                                    ; 1674  ; NOTE WELL:  We round up any odd byte count if trailer mode is used.
                                    ; 1675  ; Only PUP uses this oddity and it is a word-based, not a byte-based, protocol.
                                    ; 1676
                                    ; 1677  ;Put TDCUR on data bus and increment it
                                    ; 1678  ;R..07: J/R07,DALU_READ,ALUA/TDCUR,ALUB/TDCUR,ALUFN/ADDZA,ALUDEST/BRAMA,CRY_1
                                    ; 1679
                                    ; 1680  ;Read trailer from XPORT into buffer.  Do TDCUR-TWRAP for wrap check
                                    ; 1681  R07:    XPORT_READ,BUFR_LOAD,ALUA/TWRAP,ALUB/TDCUR,ALUFN/SUBAB,ALUDEST/NOP,
U 0426, 4272,1121,1536,3777,7760,3400  ; 1682           CRY_1
                                    ; 1683
                                    ; 1684  ;Branch on wrap check.  Compute TDCUR-TDBASE-1 (pkt) length, result to Q-reg
U 0427, 4300,0021,0477,7777,7760,3400  ; 1685          J/R07A,COND/ZERO,ALUA/TDCUR,ALUB/TDBASE,ALUFN/SUBAB,ALUDEST/Q,CRY_0
                                    ; 1686  =0
                                    ; 1687  ;No wrap. Branch on whether length is negative.  Do TEMP := TWRAP + Q-reg.
U 0430, 4340,2300,7537,7777,7760,3400  ; 1688  R07A:   J/R07C,COND/NEG,ALUA/TWRAP,ALUB/TEMP,ALUFN/ADDAQ,ALUDEST/BRAM
                                    ; 1689
                                    ; 1690  ;Wrap TDCUR, load TDBASE onto bus, then go make up for lost time.
U 0431, 4322,0247,1427,7777,7760,3400  ; 1691          J/R07B,DALU_READ,ALUA/TDBASE,ALUB/TDCUR,ALUFN/Z,ALUDEST/BRAMA
                                    ; 1692  =
                                    ; 1693  ;Compute length (TWRAP - TDBASE - 1) and write to buffer and Q-reg.
U 0432, 4332,0021,0526,7777,7760,3400  ; 1694  R07B:   BUFR_LOAD,DALU_READ,ALUA/TWRAP,ALUB/TDBASE,ALUFN/SUBAB,ALUDEST/Q,CRY_0
                                    ; 1695
                                    ; 1696  ;Do TBLEFT := TBLEFT - LENGTH - 1, join RCON 14 code.
U 0433, 4522,0320,3157,7777,7760,3400  ; 1697          J/R14D,ALUA/TBLEFT,ALUB/TBLEFT,ALUFN/SUBAQ,ALUDEST/BRAM,CRY_0
                                    ; 1698  =0
                                    ; 1699  ;Length in Q-reg is okay, copy it to TEMP and join R04 code. Put TDBASE on Dbus
U 0434, 4502,0232,7427,7777,7760,3400  ; 1700  R07C:   J/R14C,DALU_READ,ALUA/TDBASE,ALUB/TEMP,ALUFN/Q,ALUDEST/BRAMA
                                    ; 1701
                                    ; 1702  ;Length in TEMP is correct.  Put TDBASE on Dbus.
U 0435, 4362,0134,7427,7777,7760,3400  ; 1703          J/R07D,DALU_READ,ALUA/TDBASE,ALUB/TEMP,ALUFN/A,ALUDEST/NOP
                                    ; 1704  =
                                    ; 1705  ;Write the length word, then join R04 code
U 0436, 4512,0134,7766,7777,7760,3400  ; 1706  R07D:   J/R14CC,BUFR_LOAD,DALU_READ,ALUA/TEMP,ALUB/TEMP,ALUFN/A,ALUDEST/NOP
                                    ; 1707
```

```
                                    ; 1708
                                    ; 1709  ;RCON Request 12 - End Massbus Read
                                    ; 1710  ;---------------------------------
                                    ; 1711  ; 11 or 12 ins
                                    ; 1712
                                    ; 1713  ;End Massbus Read and RX packet flush differ only in the way they ack the
                                    ; 1714  ; R-Board's Demand request.  After doing the XPORT_LOAD here, we join the RX
                                    ; 1715  ; packet flush code at R04A.
                                    ; 1716
                                    ; 1717  ;Put RWBASE on DBus.
                                    ; 1718  ;R..12: J/R12,DALU_READ,ALUA/RWBASE,ALUB/TEMP,ALUFN/A,ALUDEST/NOP
                                    ; 1719
                                    ; 1720  ;Read length word into Q-reg.  Also make sure that XIPR is cleared in
                                    ; 1721  ; case we received this shutdown request before V16 completed.
U 0437, 4402,0037,7775,7777,7760,7400  ; 1722  R12:    BUFR_READ,CLEAR_XIPR,ALUA/TEMP,ALUB/TEMP,ALUFN/D,ALUDEST/Q
                                    ; 1723
                                    ; 1724  ;Branch on previous result negative.  Do RDLIMIT := -Q, just in case.
                                    ; 1725  ; Do obligatory XPORT_LOAD and join common RX buffer flush code.
                                    ; 1726          J/R04A,COND/NEG,XPORT_LOAD,ALUA/TEMP,ALUB/RDLIMIT,ALUFN/SUBZQ,
U 0440, 2760,3322,6777,5777,7760,3400  ; 1727           ALUDEST/BRAM,CRY_1
                                    ; 1728
```

```
                                    ; 1729
                                    ; 1730  ;RCON Request 14 - End Massbus Write
                                    ; 1731  ;----------------------------------
                                    ; 1732  ; 10 ins
                                    ; 1733
                                    ; 1734  ;We are joined by the trailer load code at labels R14B and R14D.
                                    ; 1735
                                    ; 1736  ;Compute TDCUR-TDBASE-1 (uncorrected pkt length) and put in Q register
                                    ; 1737  ; Do XPORT_LOAD to satisfy micro-machine.
                                    ; 1738  ;R..14: J/R14,XPORT_LOAD,ALUA/TDCUR,ALUB/TDBASE,ALUFN/SUBAB,ALUDEST/Q,CRY_0
                                    ; 1739
                                    ; 1740
                                    ; 1741  ;Branch on whether uncorrected length is negative. Do TEMP := TWRAP + Q-reg
U 0441, 4420,2300,7537,7777,7760,3400  ; 1742  R14:    J/R14AA,COND/NEG,ALUA/TWRAP,ALUB/TEMP,ALUFN/ADDAQ,ALUDEST/BRAM
                                    ; 1743
                                    ; 1744  =0
                                    ; 1745  ;Length was non-negative, so Q-reg has good length.  Branch on low byte valid.
                                    ; 1746  ;The double COND is a crock because we ran out of wires.  We ignore the
                                    ; 1747  ; TX_AVAIL_L signal and just duplicate code to handle low byte valid.
U 0442, 4441,4147,7777,7777,7760,3400  ; 1748  R14AA:  J/R14A,COND/TX_AVAIL_L.MBUS_VALID
                                    ; 1749
                                    ; 1750  ;Length was negative, so copy corrected value from TEMP to Q-reg.  Branch on
                                    ; 1751  ; low byte valid as above.
U 0443, 4441,4034,7777,7777,7760,3400  ; 1752          J/R14A,COND/TX_AVAIL_L.MBUS_VALID,ALUA/TEMP,ALUB/TEMP,ALUFN/A,ALUDEST/Q
                                    ; 1753  =
                                    ; 1754  =00
                                    ; 1755  ;Last byte is not valid.  Negate the length to TEMP, put TDBASE on bus.
U 0444, 4502,1222,7427,7777,7760,3400  ; 1756  R14A:   J/R14C,DALU_READ,ALUA/TDBASE,ALUB/TEMP,ALUFN/SUBZQ,ALUDEST/BRAMA,CRY_1
                                    ; 1757
                                    ; 1758  ;Put TDBASE on the data bus.  Put length to be written in TEMP.
U 0445, 4502,0232,7427,7777,7760,3400  ; 1759  R14B:   J/R14C,DALU_READ,ALUA/TDBASE,ALUB/TEMP,ALUFN/Q,ALUDEST/BRAMA
                                    ; 1760
                                    ; 1761  ;(Duplicate) Last byte not valid. Negate the length to TEMP, put TDBASE on bus.
U 0446, 4502,1222,7427,7777,7760,3400  ; 1762          J/R14C,DALU_READ,ALUA/TDBASE,ALUB/TEMP,ALUFN/SUBZQ,ALUDEST/BRAMA,CRY_1
                                    ; 1763
                                    ; 1764  ;(Duplicate) Put TDBASE on the data bus.  Put length to be written in TEMP.
U 0447, 4502,0232,7427,7777,7760,3400  ; 1765          J/R14C,DALU_READ,ALUA/TDBASE,ALUB/TEMP,ALUFN/Q,ALUDEST/BRAMA
                                    ; 1766  =
                                    ; 1767  ;Write length word to buffer.  Copy Q (positive length) to TEMP
U 0450, 4512,0232,7766,7777,7760,3400  ; 1768  R14C:   BUFR_LOAD,DALU_READ,ALUA/TEMP,ALUB/TEMP,ALUFN/Q,ALUDEST/BRAMA
                                    ; 1769
                                    ; 1770  ;Compute  TBLEFT := TBLEFT - LENGTH - 1.
U 0451, 4522,0311,3377,7777,7760,3400  ; 1771  R14CC:  ALUA/TEMP,ALUB/TBLEFT,ALUFN/SUBBA,ALUDEST/BRAM,CRY_0
                                    ; 1772
                                    ; 1773  ;Check if TBLEFT is now negative. Do TDBASE := TDCUR
U 0452, 4540,2334,0477,7777,7760,3400  ; 1774  R14D:   J/R14E,COND/NEG,ALUA/TDCUR,ALUB/TDBASE,ALUFN/A,ALUDEST/BRAM
                                    ; 1775
                                    ; 1776  =0
                                    ; 1777  ;Here if we must reset TX_BUFR_AVAIL_L.  Do TWRAP-TDCUR-1 for wrap check.
                                    ; 1778  R14E:   J/R14F,CLEAR_STATUS,TX_BUFR_AVAIL_L,ALUA/TWRAP,ALUB/TDCUR,ALUFN/SUBAB,
U 0454, 4532,0121,1537,7777,6760,3400  ; 1779           ALUDEST/NOP,CRY_0
                                    ; 1780
                                    ; 1781  ;Here if TBLEFT is negative.  Do TWRAP-TDCUR-1 for wrap check.
                                    ; 1782          J/R14F,SET_STATUS,TX_BUFR_AVAIL_L,ALUA/TWRAP,ALUB/TDCUR,ALUFN/SUBAB,
U 0455, 4532,0121,1537,7777,6770,3400  ; 1783           ALUDEST/NOP,CRY_0
                                    ; 1784  =
                                    ; 1785  ;Branch on above result
U 0453, 4560,0147,7777,7777,7760,3400  ; 1786  R14F:   J/R14G,COND/ZERO
                                    ; 1787
                                    ; 1788  =0
                                    ; 1789  ;Increment TDCUR and vector
U 0456, 3406,1304,1477,7777,7760,3400  ; 1790  R14G:   JUMP_VECTOR,ALUA/TDCUR,ALUB/TDCUR,ALUFN/ADDZA,ALUDEST/BRAM,CRY_1
                                    ; 1791
                                    ; 1792  ;Wrap TDCUR and vector
U 0457, 3406,0347,1477,7777,7760,3400  ; 1793          JUMP_VECTOR,ALUA/TDCUR,ALUB/TDCUR,ALUFN/Z,ALUDEST/BRAM
                                    ; 1794  =
```

; 1795

```
                                    ; 1796
                                    ; 1797  ;RCON Request 15 - Start Massbus Read
                                    ; 1798  ;-----------------------------------
                                    ; 1799  ; 7 ins.
                                    ; 1800
                                    ; 1801  ;We compute RWTOP and clear XIPR.  We don't worry about low byte valid
                                    ; 1802  ; boundary conditions when sending data to the massbus.  We do the first
                                    ; 1803  ; MPORT_LOAD here -- it is the initial 16 bits of padding.
                                    ; 1804
                                    ; 1805  ;Clear XIPR and put RWBASE on DBus.  Do obligatory XPORT_LOAD.
                                    ; 1806  ;R..15: J/R15,DALU_READ,XPORT_LOAD,CLEAR_XIPR,ALUA/RWBASE,ALUB/TEMP,ALUFN/A,
                                    ; 1807  ;        ALUDEST/NOP
                                    ; 1808
                                    ; 1809  ;Read length into Q-reg.  The MPORT_LOAD sends the 16-bits of padding to
                                    ; 1810  ; the -20 (see the monument).
U 0460, 4612,0037,7775,7377,7760,3400  ; 1811  R15:    MPORT_LOAD,BUFR_READ,ALUA/TEMP,ALUB/TEMP,ALUFN/D,ALUDEST/Q
                                    ; 1812
                                    ; 1813  ;Branch on previous result negative.  Do RWTOP := -Q, just in case.
U 0461, 4620,3322,5777,7777,7760,3400  ; 1814          J/R15A,COND/NEG,ALUA/TEMP,ALUB/RWTOP,ALUFN/SUBZQ,ALUDEST/BRAM,CRY_1
                                    ; 1815
                                    ; 1816  =0
                                    ; 1817  ;Length is positive. Do RWBASE + BUFFER[RWBASE] + 1, put result in RWTOP.
U 0462, 4642,1300,5577,7777,7760,3400  ; 1818  R15A:   J/R15B,ALUA/RWBASE,ALUB/RWTOP,ALUFN/ADDAQ,ALUDEST/BRAM,CRY_1
                                    ; 1819
                                    ; 1820  ;Length is negative (odd). Compute RWTOP as above.
U 0463, 4642,1301,5577,7777,7760,3400  ; 1821          J/R15B,ALUA/RWBASE,ALUB/RWTOP,ALUFN/ADDAB,ALUDEST/BRAM,CRY_1
                                    ; 1822  =
                                    ; 1823
                                    ; 1824  ;Do RWTOP-RWRAP to see if we need to wrap, result to Q
U 0464, 4652,1021,6277,7777,7760,3400  ; 1825  R15B:   ALUA/RWTOP,ALUB/RWRAP,ALUFN/SUBAB,ALUDEST/Q,CRY_1
                                    ; 1826
                                    ; 1827  ;Branch on wrap.  If negative, then RWTOP is okay as is. Do Q := TWRAP + Q.
U 0465, 4660,2000,7537,7777,7760,3400  ; 1828          J/R15C,COND/NEG,ALUA/TWRAP,ALUB/TEMP,ALUFN/ADDAQ,ALUDEST/Q
                                    ; 1829
                                    ; 1830  =0
                                    ; 1831  ;Do RWTOP := Q and vector
U 0466, 3406,0332,5777,7777,7760,3400  ; 1832  R15C:   JUMP_VECTOR,ALUA/TEMP,ALUB/RWTOP,ALUFN/Q,ALUDEST/BRAM
                                    ; 1833
                                    ; 1834  ;Here if RWTOP is not beyond end of buffer.  Just vector.
U 0467, 3406,0147,7777,7777,7760,3400  ; 1835          JUMP_VECTOR
                                    ; 1836  =
                                    ; 1837
```

```
                                    ; 1838
                                    ; 1839  ;RCON Request 17 - Packet status read
                                    ; 1840  ;-------------------------------------
                                    ; 1841  ; 4 or 5 ins
                                    ; 1842
                                    ; 1843  ;For the 10MB Ethernet all we return is the packet length of the current pkt.
                                    ; 1844  ;The length is stored as a word count.  If that count is positive, we add
                                    ; 1845  ; one, the double it.  If that count is negative (an odd byte count), we
                                    ; 1846  ; negate it and add one.  The extra one in both cases accounts for the extra
                                    ; 1847  ; 16 bits of initial padding we give to the user.
                                    ; 1848
                                    ; 1849  ;Put RWBASE on data bus
                                    ; 1850  ;R..17: J/R17,DALU_READ,ALUA/RWBASE,ALUB/TEMP,ALUFN/A,ALUDEST/NOP
                                    ; 1851
                                    ; 1852  ;Read length from buffer into TEMP
U 0470, 4712,0337,7775,7777,7760,3400  ; 1853  R17:    BUFR_READ,ALUA/TEMP,ALUB/TEMP,ALUFN/D,ALUDEST/BRAM
                                    ; 1854
                                    ; 1855  ;Branch on length being negative (odd count).  Do Q := TEMP + 1
U 0471, 4720,3004,7777,7777,7760,3400  ; 1856          J/R17A,COND/NEG,ALUA/TEMP,ALUB/TEMP,ALUFN/ADDZA,ALUDEST/Q,CRY_1
                                    ; 1857  =0
                                    ; 1858  ;Length is an even count. Load Xport with TEMP + Q + 1 and vector
                                    ; 1859  R17A:   JUMP_VECTOR,XPORT_LOAD,DALU_READ,ALUA/TEMP,ALUB/TEMP,ALUFN/ADDAQ,
U 0472, 3406,1100,7767,5777,7760,3400  ; 1860           ALUDEST/NOP,CRY_1
                                    ; 1861
                                    ; 1862  ;Length is odd. Make it positive.  TEMP := 0 - TEMP.
U 0473, 4742,1324,7777,7777,7760,3400  ; 1863          J/R17B,ALUA/TEMP,ALUB/TEMP,ALUFN/SUBAZ,ALUDEST/BRAM,CRY_1
                                    ; 1864  =
                                    ; 1865
                                    ; 1866  ;Load XPORT with TEMP+TEMP+1 and vector
                                    ; 1867  R17B:   JUMP_VECTOR,XPORT_LOAD,DALU_READ,ALUA/TEMP,ALUB/TEMP,ALUFN/ADDAB,
U 0474, 3406,1101,7767,5777,7760,3400  ; 1868           ALUDEST/NOP,CRY_1
                                    ; 1869
                                    ; 1870


; NUMBER OF MICRO WORDS USED:
;       D WORDS= 0
;       U WORDS= 317

        END
```

; CROSS REFERENCE LISTING

```
(U) ALUA                74 #    1623    1625    1627    1629    1631    1633    1635    1637    1639    1641    1643
                        1645    1647    1649    1651    1653
        ADDRM           89 #    1348    1402    1408
        MSIZE           93 #    397
        RDBASE          84 #    910     931     947     1032    1035    1068    1071    1154    1157    1194    1197
                        1235
        RDCUR           86 #    681     807     814     819     824     830     853     859     866     879     991
                        1055    1090    1104    1177    1217
        RDLIMIT         91 #    902     923     958     965     1012    1021    1129    1138    1510    1531    1556
        RWBASE          83 #    1340    1369    1382    1390    1453    1487    1503    1506    1534    1537    1575
                        1578    1585    1590    1818    1821
        RWCUR           85 #    694     1265    1270    1274
        RWRAP           90 #    389     391     393     431     685     906     914     927     935     955     968
                        980     998     1043    1051    1079    1086    1111    1165    1173    1206    1213    1521
                        1562
        RWTOP           87 #    1825
        SNIFF           88 #    1329    1372    1429    1447    1459    1465    1468
        TBLEFT          81 #    752     1697
        TDBASE          76 #    1691    1700    1703    1756    1759    1762    1765
        TDCUR           78 #    387     402     698     1353    1378    1685    1774    1790    1793
        TEMP            94 #    376     378     380     403     406     410     435     464     581     585     617
                        622     630     632     634     636     638     640     643     732     742     745     749
                        950     995     1001    1048    1083    1093    1108    1118    1170    1210    1220    1310
                        1325    1333    1344    1405    1491    1498    1517    1622    1624    1626    1628    1630
                        1632    1634    1636    1638    1640    1642    1644    1646    1648    1650    1652    1706
                        1722    1726    1752    1768    1771    1811    1814    1832    1853    1856    1859    1863
                        1867
        TWBASE          75 #    460     485     518     522     666     689     721
        TWCUR           77 #    382     468     496     500     532     544     547     557     564     677     711
                        716     758     763     768     776     779
        TWRAP           80 #    395     423     425     427     429     433     514     528     561     856     875
                        961     976     1039    1058    1075    1161    1180    1202    1238    1285    1300    1462
                        1513    1565    1571    1594    1681    1688    1694    1742    1778    1782    1828
        TWTOP           79 #    479     505     508
(U) ALUB                99 #
        ADDRM           115 #   1348    1402    1408
        MSIZE           119 #   395     397     433
        RDBASE          110 #   425     685     814     819     906     914     927     935     947     955     968
                        976     980     998     1043    1051    1055    1079    1086    1090    1111    1165    1173
                        1177    1206    1213    1217    1340    1537    1565    1585
        RDCUR           112 #   429     681     853     859     866     875     879     902     923     961     965
                        1058    1093    1180    1220    1235    1238
        RDLIMIT         117 #   431     824     830     856     1498    1503    1506    1517    1521    1726
        RWBASE          109 #   423     1531    1556    1562    1571
        RWCUR           111 #   427     1265    1274    1285    1534    1575    1578    1590    1594
        RWRAP           116 #   387     389     391     808     931     995     1108    1270    1459    1510    1825
        RWTOP           113 #   694     1814    1818    1821    1832
        SNIFF           114 #   1333    1453    1465    1468
        TBLEFT          107 #   435     749     752     1697    1771
        TDBASE          102 #   378     460     666     1378    1685    1694    1774
        TDCUR           104 #   382     406     698     1300    1310    1353    1681    1691    1778    1782    1790
                        1793
        TEMP            120 #   402     403     410     464     581     585     617     623     630     632     634
                        636     638     640     643     742     745     910     950     991     1001    1012    1032
                        1035    1039    1048    1068    1071    1075    1083    1104    1118    1129    1154    1157
```

; CROSS REFERENCE LISTING

```
                        1161    1170    1194    1197    1202    1210    1325    1329    1344    1369    1372    1382
                        1390    1405    1429    1447    1462    1487    1491    1513    1622    1623    1624    1625
                        1626    1627    1628    1629    1630    1631    1632    1633    1634    1635    1636    1637
                        1638    1639    1640    1641    1642    1643    1644    1645    1646    1647    1648    1649
                        1650    1651    1652    1653    1688    1700    1703    1706    1722    1742    1752    1756
                        1759    1762    1765    1768    1811    1828    1853    1856    1859    1863    1867
        TWBASE          101 #   376     485     721     776
        TWCUR           103 #   380     468     496     500     514     532     544     557     564     689     712
                        732     758     779
        TWRAP           106 #   393     547     716     764     769     958     1021    1138
        TWTOP           105 #   479     505     508     518     522     528     561     677
(U) ALUDEST             155 #
        BDOWN           161 #   387     389     391     393     395     397     582     585
        BQDOWN          160 #
        BQUP            162 #
        BRAM            159 #   376     378     380     382     423     425     427     429     431     435     464
                        479     497     501     505     508     519     523     529     533     544     557     561
                        564     617     690     732     742     749     753     776     779     853     859     867
                        875     880     961     977     991     1055    1058    1090    1093    1104    1177    1180
                        1217    1220    1235    1238    1275    1285    1310    1325    1333    1344    1349    1453
                        1465    1499    1503    1506    1517    1531    1534    1556    1571    1575    1579    1590
                        1594    1688    1697    1727    1742    1771    1774    1790    1793    1814    1818    1821
                        1832    1853    1863
        BRAMA           158 #   469     681     698     712     1266    1353    1622    1624    1626    1628    1630
                        1632    1634    1636    1638    1640    1642    1644    1646    1648    1650    1652    1691
                        1700    1756    1759    1762    1765    1768
        BUP             163 #   402     403     630     632     634     636     638     640
        NOP             157 #   407     410     461     486     515     547     623     644     667     677     685
                        694     717     722     759     764     769     808     825     831     856     902     906
                        911     914     923     927     935     947     951     955     958     965     968     980
                        995     998     1001    1012    1021    1032    1035    1040    1043    1048    1051    1068
                        1071    1076    1079    1083    1086    1108    1111    1129    1138    1154    1157    1162
                        1165    1170    1173    1194    1197    1203    1206    1210    1213    1271    1301    1329
                        1340    1369    1373    1383    1390    1402    1405    1408    1429    1447    1468    1487
                        1521    1537    1562    1565    1585    1623    1625    1627    1629    1631    1633    1635
                        1637    1639    1641    1643    1645    1647    1649    1651    1653    1681    1703    1706
                        1779    1783    1860    1868
        Q               156 #   433     745     814     819     932     1118    1378    1459    1462    1491    1510
                        1513    1685    1694    1722    1752    1811    1825    1828    1856
(U) ALUFN               125 #
        A               146 #   387     389     391     393     395     397     402     403     423     425     486
                        581     585     632     634     636     638     640     643     689     721     776     875
                        910     947     976     1032    1035    1055    1058    1068    1071    1090    1093    1154
                        1157    1177    1180    1194    1197    1217    1220    1238    1285    1329    1369    1372
                        1382    1390    1402    1405    1408    1429    1447    1468    1487    1571    1594    1623
                        1625    1627    1629    1631    1633    1635    1637    1639    1641    1643    1645    1647
                        1649    1651    1653    1703    1706    1752    1774
        ADDAB           128 #   406     518     749     1453    1506    1821    1867
        ADDAQ           127 #   752     1039    1075    1462    1503    1513    1688    1742    1818    1828    1859
        ADDDA           132 #
        ADDZA           131 #   382     427     429     468     544     564     630     681     961     991     1104
                        1235    1265    1353    1531    1534    1556    1575    1579    1590    1622    1624    1626
                        1628    1630    1632    1634    1636    1638    1640    1642    1644    1646    1648    1650
                        1652    1790    1856
        ADDZB           130 #   698     712
```

; CROSS REFERENCE LISTING

```
        ADDZQ                   129 #
        B                       145 #
        D                       147 #   464     479     505     508     742     1325    1333    1348    1491    1722    1811
                                1853
        NAANDB                  151 #   528     561
        NAANDQ                  150 #
        Q                       144 #   435     1001    1048    1083    1170    1210    1465    1517    1700    1759    1765
                                1768    1832
        SUBAB                   139 #   433     460     515     522     547     666     677     685     694     716     764
                                769     808     814     819     824     830     856     902     906     914     923     927
                                935     955     958     965     968     980     995     998     1012    1021    1043    1051
                                1079    1086    1108    1111    1129    1138    1165    1173    1206    1213    1270    1300
                                1340    1378    1459    1510    1521    1537    1562    1565    1585    1681    1685    1694
                                1778    1782    1825
        SUBAQ                   138 #   1697
        SUBAZ                   141 #   745     1863
        SUBBA                   135 #   931     1771
        SUBDA                   142 #
        SUBQA                   134 #   1161    1202
        SUBZA                   137 #   431     496     500     758     853     859     866     879     1274
        SUBZB                   136 #
        SUBZQ                   140 #   950     1118    1498    1726    1756    1762    1814
        Z                       148 #   376     378     380     410     532     557     617     623     732     779     1310
                                1344    1691    1793
(U) BUFR_LOAD                   242 #
        F                       243 #
        T                       244 #   807     950     955     1039    1048    1075    1083    1161    1170    1202    1210
                                1300    1426    1623    1625    1627    1629    1631    1633    1635    1637    1639    1641
                                1643    1645    1647    1649    1651    1653    1681    1694    1706    1768
(U) BUFR_READ                   237 #
        F                       238 #
        T                       239 #   479     505     508     716     742     763     768     1270    1432    1456    1471
                                1491    1722    1811    1853
(U) CARRY                       167 #
        F                       168 #   382     427     429     461     469     515     523     544     547     630     667
                                677     681     694     698     712     717     745     749     753     764     769     808
                                825     831     856     903     924     951     958     961     965     991     995     1012
                                1021    1104    1108    1118    1129    1138    1162    1203    1235    1266    1271    1301
                                1340    1353    1459    1499    1510    1531    1534    1537    1557    1565    1575    1579
                                1586    1591    1622    1624    1626    1628    1630    1632    1634    1636    1638    1640
                                1642    1644    1646    1648    1650    1652    1682    1727    1756    1762    1790    1814
                                1818    1821    1825    1856    1860    1863    1868
        T                       169 #   431     433     497     501     759     853     859     867     880     1275    1378
                                1685    1694    1697    1771    1779    1783
(U) CLEAR_XIPR                  310 #
        F                       311 #
        T                       312 #   1274    1382    1722
(U) COND                        50 #
        DBUS_LSB                54 #    581     592
        JUMP_ADDRS              65 #
        NEG                     52 #    514     745     758     1012    1021    1129    1138    1447    1462    1498    1513
                                1688    1726    1742    1774    1814    1828    1856
        RCON_REQ                66 #    670     673
        RUNNING.ATN_L           55 #    460     666
        RX_ENABLE_OUT.IN        53 #    496     500
```

; CROSS REFERENCE LISTING

```
        RX_EOF.LBYTE            63 #    807
        TX_AVAIL_L.MBUS_VAL     58 #    485     1748    1752
        VECTOR_REQ              67 #    437     472     475     554     557     564     588     596     599     613     626
                                643     729     732     776     853     859     863     866     875     879     976     1058
                                1093    1180    1220    1235    1238    1254    1274    1282    1285    1307    1310    1362
                                1365    1372    1386    1411    1432    1456    1471    1494    1543    1546    1575    1578
                                1598    1601    1653    1665    1790    1793    1832    1835    1859    1867
        ZERO                    51 #    468     518     522     550     711     725     772     850     871     961     972
                                991     1001    1032    1035    1068    1071    1104    1118    1154    1157    1194    1197
                                1231    1265    1278    1304    1487    1540    1556    1571    1585    1590    1594    1685
                                1786
        ZERO.FLUSH              56 #    814     819     824     830
(U) DALU_READ                   226 #
        F                       227 #
        T                       228 #   376     378     380     382     387     389     391     393     395     397     402
                                403     406     410     423     425     427     429     431     433     435     460     468
                                485     496     500     514     518     522     528     532     544     547     557     561
                                564     581     585     622     643     666     681     698     711     721     758     910
                                931     947     950     955     1032    1035    1039    1048    1068    1071    1075    1083
                                1154    1157    1161    1170    1194    1197    1202    1210    1265    1329    1353    1369
                                1372    1382    1390    1402    1405    1408    1429    1447    1453    1459    1462    1465
                                1468    1487    1622    1623    1624    1625    1626    1627    1628    1629    1630    1631
                                1632    1633    1634    1635    1636    1637    1638    1639    1640    1641    1642    1643
                                1644    1645    1646    1647    1648    1649    1650    1651    1652    1653    1691    1694
                                1700    1703    1706    1756    1759    1762    1765    1768    1859    1867
(U) J                           44 #
        IDLE                    460 #   657     658     659     660     661     662     663
        IDLE.A                  460     464 #   666
        IDLE.B                  468     479 #
        IDLE.D                  485     496 #
        IDLE.E                  479     505     508     514 #
        IDLE.F                  514     518 #
        IDLE.G                  518     522     528 #
        IDLE.H                  528     544 #
        IDLE.J                  550     554 #
        IDLE.K                  532     561 #
        INIT                    370 #   1336
        INT.A                   464     581 #
        INT.B                   581     585 #
        INT.C                   585     592 #
        INT.D                   592     596 #
        R.FIELD                 670     673     1319 #
        R00                     1325    1402 #
        R01                     1329    1426 #
        R02                     1333    1447 #
        R02A                    1447    1453 #
        R02B                    1453    1459 #
        R02C                    1462    1465 #
        R02D                    1465    1468    1471 #
        R04                     1340    1487 #
        R04A                    1498    1503 #  1726
        R04AA                   1487    1491 #
        R04AB                   1491    1498 #
        R04B                    1503    1506    1510 #
        R04C                    1513    1517 #
```

; CROSS REFERENCE LISTING

```
R04RDN                  1521    1556 #
R04RDW                  1517    1531 #
R04RNA                  1556    1562 #
R04RNB                  1565    1571 #
R04RNC                  1571    1575 #
R04RND                  1562    1585 #
R04RNE                  1585    1590 #
R04RNF                  1590    1594    1598 #
R04RWA                  1540    1543 #
R05                     1344    1622 #
R06                     1348    1665 #
R07                     1353    1681 #
R07A                    1685    1688 #
R07B                    1691    1694 #
R07C                    1688    1700 #
R07D                    1703    1706 #
R12                     1369    1722 #
R14                     1378    1742 #
R14A                    1748    1752    1756 #
R14AA                   1742    1748 #
R14B                    1759 #
R14C                    1700    1756    1759    1762    1765    1768 #
R14CC                   1706    1771 #
R14D                    1697    1774 #
R14E                    1774    1778 #
R14F                    1778    1782    1786 #
R14G                    1786    1790 #
R15                     1382    1811 #
R15A                    1814    1818 #
R15B                    1818    1821    1825 #
R15C                    1828    1832 #
R17                     1390    1853 #
R17A                    1856    1859 #
R17B                    1863    1867 #
RCV.A                   496     500     613 #
RCV.B                   622     626 #
RCV.C                   617     630 #
V12                     677     711 #
V12A                    711     716 #
V12B                    716     725 #
V12C                    725     729 #
V12D                    721     742 #
V12E                    745     749 #
V12F                    749     752     758 #
V12G                    758     763 #
V12H                    763     768     772 #
V12J                    772     776 #   779
V13                     681     807 #
V13A                    807     814 #
V13ABA                  1231    1235 #
V13ABT                  685     906     914     927     935     968     980     998     1043    1051    1079    1086
                        1111    1165    1173    1206    1213    1231 #
V13EOB                  819     923 #
V13EOF                  814     902 #
V13N2A                  1012    1068 #
```

; CROSS REFERENCE LISTING

```
        V13N2B                  1071    1075 #
        V13N2C                  1068    1083 #
        V13N2D                  1075    1083    1090 #
        V13NRA                  850     863 #
        V13NRB                  856     871 #
        V13NRC                  871     875 #
        V13NRM                  824     830     850 #
        V13NWA                  991     995 #
        V13NWB                  995     1001 #
        V13NWC                  1001    1012 #
        V13NWR                  902     991 #
        V13W2A                  1021    1032 #
        V13W2B                  1035    1039 #
        V13W2C                  1032    1048 #
        V13W2D                  1039    1048    1055 #
        V13WRA                  950     958 #
        V13WRB                  931     947 #
        V13WRC                  961     965 #
        V13WRD                  965     972 #
        V13WRE                  972     976 #
        V13WRP                  910     955 #
        V15                     689     1254 #
        V16                     694     1265 #
        V16A                    1265    1270 #
        V16B                    1270    1278 #
        V16C                    1278    1282 #
        V17                     698     1300 #
        V17A                    1304    1307 #
        V3BN2A                  1129    1194 #
        V3BN2B                  1194    1202 #
        V3BN2C                  1197    1210 #
        V3BN2D                  1202    1210    1217 #
        V3BNWA                  1104    1108 #
        V3BNWB                  1108    1118 #
        V3BNWC                  1118    1129 #
        V3BNWR                  923     1104 #
        V3BW2A                  1138    1152 #
        V3BW2B                  1154    1161 #
        V3BW2C                  1157    1170 #
        V3BW2D                  1161    1170    1177 #
        VECTOR                  437     472     475     554     557     564     588     596     599     613     626     643
                                653 #   729     732     776     853     859     863     866     875     879     976     1058
                                1093    1180    1220    1235    1238    1254    1274    1282    1285    1307    1310    1362
                                1365    1372    1386    1411    1432    1456    1471    1494    1543    1546    1575    1578
                                1598    1601    1653    1665    1790    1793    1832    1835    1859    1867
(U) LBYTE_VALID                 197 #
        F                       198 #
        T                       199 #   371     711     752
(U) MACRO%
        BUFR_LOAD               245 #   807     950     955     1039    1048    1075    1083    1161    1170    1202    1210
                                1300    1426    1623    1625    1627    1629    1631    1633    1635    1637    1639    1641
                                1643    1645    1647    1649    1651    1653    1681    1694    1706    1768
        BUFR_READ               240 #   479     505     508     716     742     763     768     1270    1432    1456    1471
                                1491    1722    1811    1853
        CLEAR_STATUS            220 #   370     415     528     532     622     689     752     763     772     1546    1578
```

; CROSS REFERENCE LISTING

```
                        1601    1778
CLEAR_XIPR              313 #   1274    1382    1722
CRY_0                   170 #   431     433     497     501     759     853     859     867     880     1275    1378
                        1685    1694    1697    1771    1779    1783
CRY_1                   171 #   382     427     429     461     469     515     523     544     547     630     667
                        677     681     694     698     712     717     745     749     753     764     769     808
                        825     831     856     903     924     951     958     961     965     991     995     1012
                        1021    1104    1108    1118    1129    1138    1162    1203    1235    1266    1271    1301
                        1340    1353    1459    1499    1510    1531    1534    1537    1557    1565    1575    1579
                        1586    1591    1622    1624    1626    1628    1630    1632    1634    1636    1638    1640
                        1642    1644    1646    1648    1650    1652    1682    1727    1756    1762    1790    1814
                        1818    1821    1825    1856    1860    1863    1868
DALU_READ               229 #   406     410     468     485     581     585     622     643     681     698     711
                        721     758     910     931     947     950     955     1032    1035    1039    1048    1068
                        1071    1075    1083    1154    1157    1161    1170    1194    1197    1202    1210    1265
                        1329    1353    1369    1372    1382    1390    1402    1405    1408    1429    1447    1465
                        1468    1487    1622    1623    1624    1625    1626    1627    1628    1629    1630    1631
                        1632    1633    1634    1635    1636    1637    1638    1639    1640    1641    1642    1643
                        1644    1645    1646    1647    1648    1649    1650    1651    1652    1653    1691    1694
                        1700    1703    1706    1756    1759    1762    1765    1768    1859    1867
DEBUG                   230 #   376     378     380     382     387     389     391     393     395     397     402
                        403     423     425     427     429     431     433     435     460     496     500     514
                        518     522     528     532     544     547     557     561     564     666     1453    1459
                        1462
DUMP_REG                1617 #  1623    1625    1627    1629    1631    1633    1635    1637    1639    1641    1643
                        1645    1647    1649    1651    1653
JUMP_VECTOR             69 #    437     472     475     554     557     564     588     596     599     613     626
                        643     729     732     776     853     859     863     866     875     879     976     1058
                        1093    1180    1220    1235    1238    1254    1274    1282    1285    1307    1310    1362
                        1365    1372    1386    1411    1432    1456    1471    1494    1543    1546    1575    1578
                        1598    1601    1653    1665    1790    1793    1832    1835    1859    1867
LBYTE_VALID             200 #   371     711     752
MPORT_LOAD              265 #   1270    1811
MPORT_READ              260 #   1300
NEXT_REG                1616 #  1622    1624    1626    1628    1630    1632    1634    1636    1638    1640    1642
                        1644    1646    1648    1650    1652
NI_BADRS_RST            273 #   370     415     617     902     910     923     931     1231
NI_COMMAND_LOAD         308 #   406     410     622     643
NI_COMMAND_READ         303 #   418     464
NI_DATA_LOAD            298 #   716     763     768
NI_DATA_READ            293 #   415     807
NI_MASK_ADRS_LOAD       278 #   1348    1402    1408
NI_MASK_DATA_LOAD       288 #   1405
NI_MASK_DATA_READ       283 #   1411    1665
NI_RX_COMMAND_LOAD      328 #   410     622     643
NI_RX_STATUS_READ       324 #
NI_TX_COMMAND_LOAD      330 #   406
NI_TX_STATUS_READ       326 #   418     464
RX_ENABLE               215 #   371     622     640
RX_OVERFLOW             185 #   371     866     879
RX_PKT_AVAIL            195 #   370     976     1055    1090    1177    1217    1546    1578    1601
SET_STATUS              221 #   514     588     599     640     711     721     768     866     879     976     1055
                        1090    1177    1217    1782
SLIDE_READ              235 #   1386
TX_ACQ_ERR              180 #   371     588     599
```

; CROSS REFERENCE LISTING

```
        TX_BUFR_AVAIL_L         190 #   371     763     768     1778    1782
        TX_EOF                  205 #   370     528     532     721
        TX_REQ_ENABLE           210 #   370     415     514     689     772
        XPORT_LOAD              255 #   1362    1365    1372    1378    1382    1386    1411    1432    1456    1471    1665
                                1726    1859    1867
        XPORT_READ              250 #   1325    1333    1336    1344    1348    1426    1494    1498    1681
(U) MPORT_LOAD                  262 #
        F                       263 #
        T                       264 #   1270    1811
(U) MPORT_READ                  257 #
        F                       258 #
        T                       259 #   1300
(U) NI_BADRS_RST                270 #
        F                       271 #
        T                       272 #   370     415     617     902     910     923     931     1231
(U) NI_COMMAND_LOAD             305 #
        F                       306 #
        T                       307 #   406     410     622     643
(U) NI_COMMAND_READ             300 #
        F                       301 #
        T                       302 #   418     464
(U) NI_DATA_LOAD                295 #
        F                       296 #
        T                       297 #   716     763     768
(U) NI_DATA_READ                290 #
        F                       291 #
        T                       292 #   415     807
(U) NI_MASK_ADRS_LOAD           275 #
        F                       276 #
        T                       277 #   1348    1402    1408
(U) NI_MASK_DATA_LOAD           285 #
        F                       286 #
        T                       287 #   1405
(U) NI_MASK_DATA_READ           280 #
        F                       281 #
        T                       282 #   1411    1665
(U) NI_REG_SEL                  319 #
        RX_REG                  321 #   410     622     643
        TX_REG                  322 #   406     418     464
(U) RX_ENABLE                   212 #
        F                       213 #
        T                       214 #   371     622     640
(U) RX_OVERFLOW                 182 #
        F                       183 #
        T                       184 #   371     866     879
(U) RX_PKT_AVAIL                192 #
        F                       193 #
        T                       194 #   370     976     1055    1090    1177    1217    1546    1578    1601
(U) SLIDE_READ                  232 #
        F                       233 #
        T                       234 #   1386
(U) STATUS                      217 #
        F                       218 #   370     415     528     532     622     689     752     763     772     1546    1578
                                1601    1778
        T                       219 #   514     588     599     640     711     721     768     866     879     976     1055
```

; CROSS REFERENCE LISTING

```
                          1090    1177    1217    1782
(U) TX_ACQ_ERR            177 #
       F                  178 #
       T                  179 #   371     588     599
(U) TX_BUFR_AVAIL_L       187 #
       F                  188 #
       T                  189 #   371     763     768     1778    1782
(U) TX_EOF                202 #
       F                  203 #
       T                  204 #   370     528     532     721
(U) TX_REQ_ENABLE         207 #
       F                  208 #
       T                  209 #   370     415     514     689     772
(U) XPORT_LOAD            252 #
       F                  253 #
       T                  254 #   1362    1365    1372    1378    1382    1386    1411    1432    1456    1471    1665
                          1726    1859    1867
(U) XPORT_READ            247 #
       F                  248 #
       T                  249 #   1325    1333    1336    1344    1348    1426    1494    1498    1681
```

```
; VEX02.MCR[,]                  MICRO 20(156)   MEIS 10MB ETHERNET DATA UCODE - GFS/KSL PAGE 60
; LOCATION / LINE NUMBER INDEX
; DCODE LOC'N   0       1       2       3       4       5       6       7

d 0000
```

; LOCATION / LINE NUMBER INDEX

; UCODE LOC'N	0	1	2	3	4	5	6	7
u 0000	371	376	378	380	382	387	389	391
u 0010	393	395	397	402	403	407	410	415
u 0020	418	423	425	427	429	431	433	435
u 0030	437	461	479	486	464	469	472	475
u 0040	497	501	505	508	515	544	519	523
u 0050	529	533	547	550	554	557	561	564
u 0060	582	592	585	588	596	599	630	632
u 0070	613	617	623	626	634	636	638	640
u 0100	644	712	717	722	725	742	729	732
u 0110	745	759	749	753	764	769	772	808
u 0120	776	779	863	867	814	819	825	831
u 0130	850	853	856	859	871	947	875	880
u 0140	903	906	911	914	924	927	932	935
u 0150	951	955	958	961	965	968	972	991
u 0160	977	980	995	998	1001	1055	1012	1021
u 0170	1032	1035	1040	1043	1048	1051	1058	1090
u 0200	1068	1071	1076	1079	1083	1086	1093	1104
u 0210	1108	1111	1118	1177	1129	1138	1154	1157
u 0220	1162	1165	1170	1173	1180	1217	1194	1197
u 0230	1203	1206	1210	1213	1220	1231	1235	1238
u 0240	1254	1266	1271	1275	1278	1301	1282	1285
u 0250	1304	1402	1307	1310	1405	1408	1411	1426
u 0260	1429	1432	1447	1459	1453	1456	1462	1471
u 0270	1465	1468	1487	1499	1491	1494	1503	1506
u 0300	1510	1513	1517	1521	1531	1534	1537	1540
u 0310	1543	1546	1557	1571	1562	1565	1575	1579
u 0320	1586	1622	1591	1594	1598	1601	1623	1624
u 0330	1625	1626	1627	1628	1629	1630	1631	1632
u 0340	657	658	659	660	661	662	663	667
u 0350	670	673	677	681	685	690	694	698
u 0360	1325	1329	1333	1336	1340	1344	1349	1353
u 0370	1362	1365	1369	1373	1378	1383	1386	1390
u 0400	1633	1634	1635	1636	1637	1638	1639	1640
u 0410	1641	1642	1643	1644	1645	1646	1647	1648
u 0420	1649	1650	1651	1652	1653	1665	1682	1685
u 0430	1688	1691	1694	1697	1700	1703	1706	1722
u 0440	1727	1742	1748	1752	1756	1759	1762	1765
u 0450	1768	1771	1774	1786	1779	1783	1790	1793
u 0460	1811	1814	1818	1821	1825	1828	1832	1835
u 0470	1853	1856	1860	1863	1868			

no errors detected
END OF MICRO CODE ASSEMBLY
used 5.94 seconds