```
0001   0000             ;ROM system monitor
0002   0000             ;Macro definitions for three levels of nested call and ret
0003   0000             #define call0(address) \return: .set $+15\ ldm return\ stm return_jump0+1\ ldm return+1\ stm return_jump0+2
0004   0000             #defcont \ jmp address\ .dw $+2
0005   0000             #define ret0 \ jmp return_jump0
0006   0000             #define call1(address) \return: .set $+15\ ldm return\ stm return_jump1+1\ ldm return+1\ stm return_jump1+2
0007   0000             #defcont \ jmp address\ .dw $+2
0008   0000             #define ret1 \ jmp return_jump1
0009   0000             #define call2(address) \return: .set $+15\ ldm return\ stm return_jump2+1\ ldm return+1\ stm return_jump2+2
0010   0000             #defcont \ jmp address\ .dw $+2
0011   0000             #define ret2 \ jmp return_jump2
0012   0000
0013   0000             ;Buffer location defined by these constant values
0014   0000             ;Needs to be in RAM above variables and variable instructions
0015   0000             buff_low:    .equ    80h     ;low byte of buffer address
0016   0000             buff_high:   .equ    08h     ;high byte of buffer address
0017   0000             buffer:              .equ    0880h   ;two-byte address constant
0018   0000                          .org    0000h
0019   0000
0020   0000 1F          return:              .db     1fh     ;placeholder for first definition of variable label -- NOP
0021   0001
0022   0001             ;Initialize port
0023   0001 10 4E                    LDI     4EH     ;1 stop bit, no parity, 8-bit char, 16x baud
0024   0003 18 03                    OUT     03H     ;write to UART control port
0025   0005 10 37                    LDI     37H     ;enable receive and transmit
0026   0007 18 03                    OUT     03H     ;write to control port
0027   0009
0028   0009             ;Opcode initialization for RAM instructions
0029   0009 10 13                    ldi     13h                     ;jmp opcode
0030   000B 12 1E 08                 stm     ld_indexed_stm+3        ;return jumps for indexed instructions
0031   000E 12 24 08                 stm     d_indexed_ldm+3
0032   0011 12 2A 08                 stm     d_indexed_stm+3
0033   0014 12 30 08                 stm     bl_indexed_stm+3
0034   0017 12 36 08                 stm     ws_inst+3
0035   001A 12 3C 08                 stm     gl_indexed_stm+3
0036   001D 12 3F 08                 stm     return_jump0            ;other variable jumps
0037   0020 12 42 08                 stm     return_jump1
0038   0023 12 45 08                 stm     return_jump2
0039   0026 12 18 08                 stm     run_jump
0040   0029 10 12                    ldi     12h                     ;stm opcode
0041   002B 12 1B 08                 stm     ld_indexed_stm          ;indexed store instructions
0042   002E 12 27 08                 stm     d_indexed_stm
0043   0031 12 2D 08                 stm     bl_indexed_stm
0044   0034 12 39 08                 stm     gl_indexed_stm
0045   0037 10 11                    ldi     11h                     ;ldm opcode
0046   0039 12 21 08                 stm     d_indexed_ldm
0047   003C 12 33 08                 stm     ws_inst
0048   003F 10 D3                    ldi     0d3h                    ;address of ld_stm_back
0049   0041 12 1F 08                 stm     ld_indexed_stm+4
0050   0044 10 02                    ldi     02h
0051   0046 12 20 08                 stm     ld_indexed_stm+5
```

```
0052   0049 10 59                  ldi     59h                 ;address of d_ldm_back
0053   004B 12 25 08               stm     d_indexed_ldm+4
0054   004E 10 01                  ldi     01h
0055   0050 12 26 08               stm     d_indexed_ldm+5
0056   0053 10 78                  ldi     78h                 ;address of d_stm_back
0057   0055 12 2B 08               stm     d_indexed_stm+4
0058   0058 10 01                  ldi     01h
0059   005A 12 2C 08               stm     d_indexed_stm+5
0060   005D 10 8A                  ldi     8ah                 ;address of bl_back
0061   005F 12 31 08               stm     bl_indexed_stm+4
0062   0062 10 04                  ldi     04h
0063   0064 12 32 08               stm     bl_indexed_stm+5
0064   0067 10 C9                  ldi     0c9h                ;address of ws_back
0065   0069 12 37 08               stm     ws_inst+4
0066   006C 10 05                  ldi     05h
0067   006E 12 38 08               stm     ws_inst+5
0068   0071 10 98                  ldi     98h                 ;address of gl_back
0069   0073 12 3D 08               stm     gl_indexed_stm+4
0070   0076 10 05                  ldi     05h
0071   0078 12 3E 08               stm     gl_indexed_stm+5
0072   007B
0073   007B             ;Print greeting
0074   007B 11 0E 07    sm_lukewarm ldm     sm_greeting
0075   007E 12 34 08               stm     ws_inst+1
0076   0081 11 0F 07               ldm     sm_greeting+1
0077   0084 12 35 08               stm     ws_inst+2
0078   0087                        call0(write_string)
0078   0087
0078   0087 11 96 00
0078   008A 12 40 08
0078   008D 11 97 00
0078   0090 12 41 08
0078   0093 13 BF 05
0078   0096 98 00
0079   0098
0080   0098             ;Warm start for system monitor, re-entry point after commands have finished
0081   0098             ;Prompt for routine number input
0082   0098 11 40 07    sm_warm             ldm     sm_prompt
0083   009B 12 34 08               stm     ws_inst+1
0084   009E 11 41 07               ldm     sm_prompt+1
0085   00A1 12 35 08               stm     ws_inst+2
0086   00A4                        call0(write_string)
0086   00A4
0086   00A4 11 B3 00
0086   00A7 12 40 08
0086   00AA 11 B4 00
0086   00AD 12 41 08
0086   00B0 13 BF 05
0086   00B3 B5 00
0087   00B5
0088   00B5             ;Get character and jump to monitor routine
```

```
0089   00B5 17 03       sm_chk_loop: in      03h            ;get status
0090   00B7 0C 02                    andim   02h            ;check RxRDY
0091   00B9 14 B5 00                 jpz     sm_chk_loop
0092   00BC 17 02                    in      02h            ;get char from port and echo
0093   00BE 12 11 08                 stm     choice
0094   00C1 17 03       sm_echo_loop in      03h
0095   00C3 0C 01                    andim   01h            ;check TxRDY
0096   00C5 14 C1 00                 jpz     sm_echo_loop
0097   00C8 11 11 08                 ldm     choice
0098   00CB 18 02                    out     02h
0099   00CD 0F 35                    cmp     '5'
0100   00CF 16 ED 02                 jpc     sm_bload
0101   00D2 0F 34                    cmp     '4'
0102   00D4 16 09 02                 jpc     sm_load
0103   00D7 0F 33                    cmp     '3'
0104   00D9 16 F7 01                 jpc     sm_run
0105   00DC 0F 32                    cmp     '2'
0106   00DE 16 E4 00                 jpc     sm_dump
0107   00E1 13 98 00                 jmp     sm_warm        ;any number other than 2 to 5 results in warm restart
0108   00E4
0109   00E4             ;Memory dump routine
0110   00E4             ;Get address from input string
0111   00E4 13 BA 04    sm_dump              jmp    get_address
0112   00E7 11 0E 08    d_addr_back  ldm     address
0113   00EA 12 22 08                 stm     d_indexed_ldm+1
0114   00ED 11 0F 08                 ldm     address+1
0115   00F0 12 23 08                 stm     d_indexed_ldm+2
0116   00F3
0117   00F3             ;Dump 16 lines of 16 characters each
0118   00F3             ;Set up line counter
0119   00F3 10 10                    ldi     16
0120   00F5 12 14 08                 stm     line_counter
0121   00F8             ;Loop for putting a memory dump line in the buffer
0122   00F8             ;Start with 4 characters of the starting address of the line, followed by space
0123   00F8 11 23 08    d_line_loop  ldm     d_indexed_ldm+2       ;high byte of memory address
0124   00FB 12 12 08                 stm     byte
0125   00FE                          call0(byte_to_hex_pair)
0125   00FE
0125   00FE 11 0D 01
0125   0101 12 40 08
0125   0104 11 0E 01
0125   0107 12 41 08
0125   010A 13 8C 06
0125   010D 0F 01
0126   010F 11 0A 08                 ldm     char_pair
0127   0112 12 80 08                 stm     buffer         ;start of line
0128   0115 11 0B 08                 ldm     char_pair+1
0129   0118 12 81 08                 stm     buffer+1
0130   011B 11 22 08                 ldm     d_indexed_ldm+1       ;low byte of memory address
0131   011E 12 12 08                 stm     byte
0132   0121                          call0(byte_to_hex_pair)
```

```
0132   0121
0132   0121 11 30 01
0132   0124 12 40 08
0132   0127 11 31 01
0132   012A 12 41 08
0132   012D 13 8C 06
0132   0130 32 01
0133   0132 11 0A 08                ldm    char_pair
0134   0135 12 82 08                stm    buffer+2
0135   0138 11 0B 08                ldm    char_pair+1
0136   013B 12 83 08                stm    buffer+3
0137   013E 10 20                   ldi    20h            ;space character
0138   0140 12 84 08                stm    buffer+4
0139   0143            ;Set up for getting 16 memory bytes, converting to characters, and putting in string buffer
0140   0143 10 80                   ldi    buff_low
0141   0145 08 05                   addim  5
0142   0147 12 28 08                stm    d_indexed_stm+1       ;low byte of location of first character in output string
0143   014A 10 08                   ldi    buff_high
0144   014C 09 00                   adcim  0              ;16-bit addition
0145   014E 12 29 08                stm    d_indexed_stm+2       ;high byte of location of first character in output string
0146   0151 10 10                   ldi    16
0147   0153 12 16 08                stm    byte_counter   ;number of bytes to get, convert, and display in one line
0148   0156 13 21 08    d_byte_loop: jmp    d_indexed_ldm  ;get byte from memory
0149   0159 12 12 08    d_ldm_back:  stm    byte
0150   015C                         call0(byte_to_hex_pair)      ;convert to hex pair
0150   015C
0150   015C 11 6B 01
0150   015F 12 40 08
0150   0162 11 6C 01
0150   0165 12 41 08
0150   0168 13 8C 06
0150   016B 6D 01
0151   016D 10 03                   ldi    3
0152   016F 12 17 08                stm    nybble_counter
0153   0172 11 0A 08                ldm    char_pair
0154   0175 13 27 08    d_nybble_loop:      jmp    d_indexed_stm  ;store char of byte in string buffer
0155   0178 11 28 08    d_stm_back:  ldm    d_indexed_stm+1       ;increment pointer by 16-bit incrementation
0156   017B 19                      inc
0157   017C 12 28 08                stm    d_indexed_stm+1
0158   017F 11 29 08                ldm    d_indexed_stm+2
0159   0182 09 00                   adcim  0
0160   0184 12 29 08                stm    d_indexed_stm+2       ;pointing to next spot in buffer
0161   0187 11 17 08                ldm    nybble_counter
0162   018A 1A                      dec                   ;all three characters stored (hex chars plus space)?
0163   018B 14 A1 01                jpz    d_nybble_done  ;yes, next byte
0164   018E 12 17 08                stm    nybble_counter ;no, place next character or space
0165   0191 0A 01                   subim  1              ;if nybble count = 1, put a space next
0166   0193 14 9C 01                jpz    d_put_space
0167   0196 11 0B 08                ldm    char_pair+1    ;otherwise, get next char and store
0168   0199 13 75 01                jmp    d_nybble_loop
0169   019C 10 20       d_put_space: ldi    20h            ;space character
```

```
0170   019E 13 75 01                jmp    d_nybble_loop
0171   01A1 11 22 08    d_nybble_done:     ldm    d_indexed_ldm+1       ;increment memory pointer
0172   01A4 19                      inc
0173   01A5 12 22 08                stm    d_indexed_ldm+1
0174   01A8 11 23 08                ldm    d_indexed_ldm+2
0175   01AB 09 00                   adcim  0
0176   01AD 12 23 08                stm    d_indexed_ldm+2
0177   01B0 11 16 08                ldm    byte_counter
0178   01B3 1A                      dec
0179   01B4 14 BD 01                jpz    d_line_done
0180   01B7 12 16 08                stm    byte_counter
0181   01BA 13 56 01                jmp    d_byte_loop
0182   01BD 10 0D       d_line_done: ldi    0dh           ;newline characters
0183   01BF 12 B4 08                stm    buffer+52
0184   01C2 10 0A                   ldi    0ah
0185   01C4 12 B5 08                stm    buffer+53
0186   01C7 10 00                   ldi    0
0187   01C9 12 B6 08                stm    buffer+54     ;where the end of the line will be
0188   01CC             ;Write string to screen
0189   01CC 10 80                   ldi    buff_low
0190   01CE 12 34 08                stm    ws_inst+1
0191   01D1 10 08                   ldi    buff_high
0192   01D3 12 35 08                stm    ws_inst+2
0193   01D6                         call0(write_string)
0193   01D6
0193   01D6 11 E5 01
0193   01D9 12 40 08
0193   01DC 11 E6 01
0193   01DF 12 41 08
0193   01E2 13 BF 05
0193   01E5 E7 01
0194   01E7
0195   01E7             ;Check if 16 lines done
0196   01E7 11 14 08                ldm    line_counter
0197   01EA 1A                      dec
0198   01EB 14 F4 01                jpz    d_done
0199   01EE 12 14 08                stm    line_counter
0200   01F1 13 F8 00                jmp    d_line_loop
0201   01F4             d_done:
0202   01F4 13 98 00    error:              jmp    sm_warm
0203   01F7
0204   01F7             ;Monitor routine to jump and execute code
0205   01F7             ;Gets target address from terminal
0206   01F7
0207   01F7 13 BA 04    sm_run              jmp    get_address
0208   01FA 11 0E 08    run_addr_back       ldm    address
0209   01FD 12 19 08                stm    run_jump+1
0210   0200 11 0F 08                ldm    address+1
0211   0203 12 1A 08                stm    run_jump+2
0212   0206 13 18 08                jmp    run_jump
0213   0209
```

```
0214   0209             ;Routine to get hex char pairs from input and load bytes in RAM
0215   0209             ;Get address first
0216   0209 13 BA 04    sm_load             jmp    get_address
0217   020C 11 0E 08    ld_addr_back ldm    address
0218   020F 12 1C 08                stm    ld_indexed_stm+1
0219   0212 11 0F 08                ldm    address+1
0220   0215 12 1D 08                stm    ld_indexed_stm+2
0221   0218             ;Get characters
0222   0218             ;First character of pair
0223   0218 17 03       ld_get_hi:   IN     03H            ;Get hi-order nybble of pair
0224   021A 0C 02                    ANDIM  02H            ;check RxRDY bit
0225   021C 14 18 02                JPZ    ld_get_hi             ;not ready, loop
0226   021F 17 02                   IN     02H            ;get char from data port
0227   0221 12 10 08                STM    temp           ;Store character
0228   0224 0A 0D                   SUBIM  0DH            ;Carriage return?
0229   0226 14 E5 02                JPZ    ld_done        ;Yes, return to monitor
0230   0229 17 03       ld_loop_1:   IN     03H            ;No, output character and validate
0231   022B 0C 01                    ANDIM  01H            ;check TxRDY bit
0232   022D 14 29 02                JPZ    ld_loop_1      ;loop if not ready
0233   0230 11 10 08                LDM    temp           ;get char back
0234   0233 18 02                   OUT    02H            ;send to UART for output
0235   0235             ;Code to validate hex character
0236   0235 0F 30                   CMP    30H            ;Lower limit of hex characters
0237   0237 16 3D 02                JPC    ld_next_1      ;Char >= 30H, possibly valid
0238   023A 13 5B 02                JMP    ld_invalid     ;Char < 30H, invalid hex char
0239   023D 0F 47       ld_next_1:   CMP    47H            ;ASCII for "G"
0240   023F 16 5B 02                JPC    ld_invalid     ;Char is G or greater, invalid
0241   0242 0F 41                   CMP    41H            ;ASCII for "A"
0242   0244 16 4F 02                JPC    ld_validAF_hi  ;Char is valid A-F
0243   0247 0F 3A                   CMP    3AH            ;ASCII for ":"
0244   0249 16 5B 02                JPC    ld_invalid     ;Char is ":" or greater, but < "A", invalid
0245   024C 13 56 02                JMP    ld_valid09_hi  ;Char is valid 0-9
0246   024F 0C 0F       ld_validAF_hi:      ANDIM  0FH            ;Mask off high bits
0247   0251 08 09                   ADDIM  9              ;Adjust ASCII to binary value
0248   0253 13 5E 02                JMP    ld_shift_hi
0249   0256 0C 0F       ld_valid09_hi:      ANDIM  0FH            ;Mask off high bits
0250   0258 13 5E 02                JMP    ld_shift_hi
0251   025B 13 E8 02    ld_invalid:  JMP    ld_error       ;Invalid hex char, quit
0252   025E 12 12 08    ld_shift_hi: STM    byte           ;Will eventually contain the byte to load
0253   0261 12 10 08                STM    temp           ;Value to add
0254   0264 10 10                   LDI    10H            ;Multiply x 16 to shift into high-order nybble
0255   0266 12 13 08                STM    counter
0256   0269 11 13 08    ld_multloop:        LDM    counter
0257   026C 1A                      DEC
0258   026D 14 7F 02                JPZ    ld_get_lo      ;Have added 16 times, done
0259   0270 12 13 08                STM    counter
0260   0273 11 10 08                LDM    temp           ;Original nybble
0261   0276 00 12 08                ADD    byte           ;Add to BYTE and store
0262   0279 12 12 08                STM    byte
0263   027C 13 69 02                JMP    ld_multloop    ;Keep adding
0264   027F 17 03       ld_get_lo:   IN     03H            ;Get lo-order nybble of pair
```

```
0265   0281 0C 02                ANDIM  02H           ;check RxRDY bit
0266   0283 14 7F 02             JPZ    ld_get_lo     ;not ready, loop
0267   0286 17 02                IN     02H           ;get char from data port
0268   0288 12 10 08             STM    temp          ;Store character
0269   028B 17 03       ld_loop2:    IN     03H           ;Output character
0270   028D 0C 01                ANDIM  01H           ;check TxRDY bit
0271   028F 14 8B 02             JPZ    ld_loop2
0272   0292 11 10 08             LDM    temp          ;When ready, retrieve character and output
0273   0295 18 02                OUT    02H
0274   0297 17 03       ld_loop3:    IN     03H
0275   0299 0C 01                ANDIM  01H
0276   029B 14 97 02             JPZ    ld_loop3
0277   029E 10 20                LDI    20H           ;Space character
0278   02A0 18 02                OUT    02H           ;send to UART for output
0279   02A2              ;Code to validate hex character
0280   02A2 11 10 08             LDM    temp          ;Retrieve character and validate
0281   02A5 0F 30                CMP    30H           ;Lower limit of hex characters
0282   02A7 16 AD 02             JPC    ld_next2             ;Char >= 30H, possibly valid
0283   02AA 13 5B 02             JMP    ld_invalid    ;Char < 30H, invalid hex char
0284   02AD 0F 47       ld_next2:    CMP    47H           ;ASCII for "G"
0285   02AF 16 5B 02             JPC    ld_invalid    ;Char is G or greater, invalid
0286   02B2 0F 41                CMP    41H           ;ASCII for "A"
0287   02B4 16 BF 02             JPC    ld_validAF_lo ;Char is valid A-F
0288   02B7 0F 3A                CMP    3AH           ;ASCII for ":"
0289   02B9 16 5B 02             JPC    ld_invalid    ;Char is ":" or greater, but < "A", invalid
0290   02BC 13 C9 02             JMP    ld_valid09_lo ;Char is valid 0-9
0291   02BF 0C 0F       ld_validAF_lo:      ANDIM  0FH           ;Mask off high bits
0292   02C1 08 09                ADDIM  9             ;Now lo nybble correct
0293   02C3 00 12 08             ADD    byte          ;Combine with hi nybble stored in BYTE
0294   02C6 13 1B 08             JMP    ld_indexed_stm ;Store the byte in RAM
0295   02C9 0C 0F       ld_valid09_lo:      ANDIM  0FH           ;Mask off high bits
0296   02CB 00 12 08             ADD    byte          ;Now full byte assembled
0297   02CE 18 00                OUT    00H           ;Display on LEDs
0298   02D0 13 1B 08             JMP    ld_indexed_stm ;Store the byte in RAM
0299   02D3 11 1C 08    ld_stm_back: LDM    ld_indexed_stm+1      ;Increment byte pointer, lo byte first
0300   02D6 19                   INC
0301   02D7 12 1C 08             STM    ld_indexed_stm+1
0302   02DA 11 1D 08             LDM    ld_indexed_stm+2      ;Increment hi byte if a carry occurred when lo byte incremented
0303   02DD 09 00                ADCIM  00H
0304   02DF 12 1D 08             STM    ld_indexed_stm+2
0305   02E2 13 18 02             JMP    ld_get_hi
0306   02E5 13 98 00    ld_done:     JMP    sm_warm       ;Return to monitor
0307   02E8 18 00       ld_error:    OUT    00H           ;Display erroneous character on LEDs
0308   02EA 13 98 00             jmp    sm_warm
0309   02ED
0310   02ED              ;Monitor routine for binary load
0311   02ED              ;Gets load target address and number of bytes from terminal
0312   02ED              ;Loads bytes in RAM and returns to monitor
0313   02ED 13 BA 04    sm_bload     jmp    get_address
0314   02F0 11 0E 08    bl_addr_back ldm    address
0315   02F3 12 2E 08                 stm    bl_indexed_stm+1
```

```
0316   02F6 11 0F 08                ldm    address+1
0317   02F9 12 2F 08                stm    bl_indexed_stm+2
0318   02FC
0319   02FC             ;Gets number of bytes in decimal from input using get_line, called as level 0 subroutine
0320   02FC             ;Write newline
0321   02FC 11 8D 07    bl_get_bytes ldm    bytes_str
0322   02FF 12 34 08                stm    ws_inst+1
0323   0302 11 8E 07                ldm    bytes_str+1
0324   0305 12 35 08                stm    ws_inst+2
0325   0308                         call0(write_string)
0325   0308
0325   0308 11 17 03
0325   030B 12 40 08
0325   030E 11 18 03
0325   0311 12 41 08
0325   0314 13 BF 05
0325   0317 19 03
0326   0319
0327   0319             ;Get input decimal number string
0328   0319                         call0(get_line)
0328   0319
0328   0319 11 28 03
0328   031C 12 40 08
0328   031F 11 29 03
0328   0322 12 41 08
0328   0325 13 65 05
0328   0328 2A 03
0329   032A
0330   032A             ;Get word value from input string
0331   032A             ;No error checking for final value -- must be between 0 and 65535 (0000 and FFFF hex)
0332   032A             ;No error checking for numerals -- must be 0 to 9
0333   032A
0334   032A 10 00                   ldi    0              ;starting value
0335   032C 12 00 08                stm    dp_value       ;zero final value variable
0336   032F 12 01 08                stm    dp_value+1
0337   0332 11 07 08    dp_input    ldm    gl_str_len     ;input string length from get_line
0338   0335 0F 05                   cmp    5
0339   0337 16 51 03                jpc    dp_setup_5
0340   033A 0F 04                   cmp    4
0341   033C 16 72 03                jpc    dp_setup_4
0342   033F 0F 03                   cmp    3
0343   0341 16 8D 03                jpc    dp_setup_3
0344   0344 0F 02                   cmp    2
0345   0346 16 A2 03                jpc    dp_setup_2
0346   0349 0F 01                   cmp    1
0347   034B 16 B1 03                jpc    dp_setup_1
0348   034E 13 98 00                jmp    sm_warm
0349   0351
0350   0351 11 84 08    dp_setup_5  ldm    buffer+4
0351   0354 12 06 08                stm    dp_1s
0352   0357 11 83 08                ldm    buffer+3
```

```
0353   035A 12 05 08                stm    dp_10s
0354   035D 11 82 08                ldm    buffer+2
0355   0360 12 04 08                stm    dp_100s
0356   0363 11 81 08                ldm    buffer+1
0357   0366 12 03 08                stm    dp_1000s
0358   0369 11 80 08                ldm    buffer
0359   036C 12 02 08                stm    dp_10000s
0360   036F 13 BA 03                jmp    dp_10000_mult
0361   0372 11 83 08    dp_setup_4  ldm    buffer+3
0362   0375 12 06 08                stm    dp_1s
0363   0378 11 82 08                ldm    buffer+2
0364   037B 12 05 08                stm    dp_10s
0365   037E 11 81 08                ldm    buffer+1
0366   0381 12 04 08                stm    dp_100s
0367   0384 11 80 08                ldm    buffer
0368   0387 12 03 08                stm    dp_1000s
0369   038A 13 DC 03                jmp    dp_1000_mult
0370   038D 11 82 08    dp_setup_3  ldm    buffer+2
0371   0390 12 06 08                stm    dp_1s
0372   0393 11 81 08                ldm    buffer+1
0373   0396 12 05 08                stm    dp_10s
0374   0399 11 80 08                ldm    buffer
0375   039C 12 04 08                stm    dp_100s
0376   039F 13 FE 03                jmp    dp_100_mult
0377   03A2 11 81 08    dp_setup_2  ldm    buffer+1
0378   03A5 12 06 08                stm    dp_1s
0379   03A8 11 80 08                ldm    buffer
0380   03AB 12 05 08                stm    dp_10s
0381   03AE 13 20 04                jmp    dp_10_mult
0382   03B1 11 80 08    dp_setup_1  ldm    buffer
0383   03B4 12 06 08                stm    dp_1s
0384   03B7 13 42 04                jmp    dp_1_mult
0385   03BA
0386   03BA
0387   03BA
0388   03BA             ;decimal parser multiplication
0389   03BA 11 02 08    dp_10000_mult       ldm    dp_10000s
0390   03BD 0A 30                   subim  30h            ;ASCII for '0'
0391   03BF 14 DC 03                jpz    dp_10000_done
0392   03C2 10 10                   ldi    10h            ;hex low byte of 10,000 decimal
0393   03C4 00 00 08                addm   dp_value
0394   03C7 12 00 08                stm    dp_value
0395   03CA 10 27                   ldi    27h            ;hex high byte of 10,000 decimal
0396   03CC 01 01 08                adcm   dp_value+1
0397   03CF 12 01 08                stm    dp_value+1
0398   03D2 11 02 08                ldm    dp_10000s
0399   03D5 1A                      dec
0400   03D6 12 02 08                stm    dp_10000s
0401   03D9 13 BA 03                jmp    dp_10000_mult
0402   03DC             dp_10000_done
0403   03DC 11 03 08    dp_1000_mult ldm    dp_1000s
```

```
0404   03DF 0A 30                 subim  30h           ;ASCII for '0'
0405   03E1 14 FE 03              jpz    dp_1000_done
0406   03E4 10 E8                 ldi    0e8h          ;hex low byte of 1000 decimal
0407   03E6 00 00 08              addm   dp_value
0408   03E9 12 00 08              stm    dp_value
0409   03EC 10 03                 ldi    03h           ;hex high byte of 1000 decimal
0410   03EE 01 01 08              adcm   dp_value+1
0411   03F1 12 01 08              stm    dp_value+1
0412   03F4 11 03 08              ldm    dp_1000s
0413   03F7 1A                    dec
0414   03F8 12 03 08              stm    dp_1000s
0415   03FB 13 DC 03              jmp    dp_1000_mult
0416   03FE             dp_1000_done
0417   03FE 11 04 08    dp_100_mult  ldm    dp_100s
0418   0401 0A 30                 subim  30h           ;ASCII for '0'
0419   0403 14 20 04              jpz    dp_100_done
0420   0406 10 64                 ldi    64h           ;hex low byte of 100 decimal
0421   0408 00 00 08              addm   dp_value
0422   040B 12 00 08              stm    dp_value
0423   040E 10 00                 ldi    00h           ;hex high byte of 100 decimal
0424   0410 01 01 08              adcm   dp_value+1
0425   0413 12 01 08              stm    dp_value+1
0426   0416 11 04 08              ldm    dp_100s
0427   0419 1A                    dec
0428   041A 12 04 08              stm    dp_100s
0429   041D 13 FE 03              jmp    dp_100_mult
0430   0420             dp_100_done
0431   0420 11 05 08    dp_10_mult   ldm    dp_10s
0432   0423 0A 30                 subim  30h           ;ASCII for '0'
0433   0425 14 42 04              jpz    dp_10_done
0434   0428 10 0A                 ldi    0ah           ;hex low byte of 10 decimal
0435   042A 00 00 08              addm   dp_value
0436   042D 12 00 08              stm    dp_value
0437   0430 10 00                 ldi    00h           ;hex high byte of 10 decimal
0438   0432 01 01 08              adcm   dp_value+1
0439   0435 12 01 08              stm    dp_value+1
0440   0438 11 05 08              ldm    dp_10s
0441   043B 1A                    dec
0442   043C 12 05 08              stm    dp_10s
0443   043F 13 20 04              jmp    dp_10_mult
0444   0442             dp_10_done
0445   0442 11 06 08    dp_1_mult    ldm    dp_1s
0446   0445 0A 30                 subim  30h           ;ASCII for '0'
0447   0447 00 00 08              addm   dp_value
0448   044A 12 00 08              stm    dp_value
0449   044D 10 00                 ldi    00h           ;hex high byte of 100 decimal
0450   044F 01 01 08              adcm   dp_value+1
0451   0452 12 01 08              stm    dp_value+1
0452   0455
0453   0455             ;Set up byte counter and write ready string
0454   0455 11 00 08              ldm    dp_value
```

```
0455   0458 12 08 08                stm     bl_byte_counter
0456   045B 11 01 08                ldm     dp_value+1
0457   045E 12 09 08                stm     bl_byte_counter+1
0458   0461 11 A7 07                ldm     bl_ready_str
0459   0464 12 34 08                stm     ws_inst+1
0460   0467 11 A8 07                ldm     bl_ready_str+1
0461   046A 12 35 08                stm     ws_inst+2
0462   046D                         call0(write_string)
0462   046D
0462   046D 11 7C 04
0462   0470 12 40 08
0462   0473 11 7D 04
0462   0476 12 41 08
0462   0479 13 BF 05
0462   047C 7E 04
0463   047E
0464   047E             ;Loop to get binary data and store
0465   047E 17 03       bl_chk_loop: in      03h            ;get status
0466   0480 0C 02                    andim   02h            ;check RxRDY
0467   0482 14 7E 04                 jpz     bl_chk_loop
0468   0485 17 02                    in      02h            ;get binary from port
0469   0487 13 2D 08                 jmp     bl_indexed_stm ;store in RAM
0470   048A 11 2E 08    bl_back              ldm     bl_indexed_stm+1;increment pointer
0471   048D 19                       inc
0472   048E 12 2E 08                 stm     bl_indexed_stm+1
0473   0491 11 2F 08                 ldm     bl_indexed_stm+2
0474   0494 09 00                    adcim   0
0475   0496 12 2F 08                 stm     bl_indexed_stm+2
0476   0499 11 08 08                 ldm     bl_byte_counter       ;decrement byte counter
0477   049C 1A                       dec
0478   049D 12 08 08                 stm     bl_byte_counter
0479   04A0 11 09 08                 ldm     bl_byte_counter+1
0480   04A3 0B 00                    sbbim   0
0481   04A5 12 09 08                 stm     bl_byte_counter+1
0482   04A8 14 AE 04                 jpz     bl_low_zero    ;check if byte counter = zero
0483   04AB 13 7E 04                 jmp     bl_chk_loop
0484   04AE 11 08 08    bl_low_zero  ldm     bl_byte_counter
0485   04B1 14 B7 04                 jpz     bl_done        ;yes, done -- return to monitor
0486   04B4 13 7E 04                 jmp     bl_chk_loop    ;no, get next byte
0487   04B7 13 98 00    bl_done              jmp     sm_warm
0488   04BA
0489   04BA
0490   04BA
0491   04BA             ;Routine to get address
0492   04BA             ;Not called as a subroutine, return jump by switch structure
0493   04BA 11 79 07    get_address  ldm     addr_str
0494   04BD 12 34 08                 stm     ws_inst+1
0495   04C0 11 7A 07                 ldm     addr_str+1
0496   04C3 12 35 08                 stm     ws_inst+2
0497   04C6                          call0(write_string)
0497   04C6
```

```
0497   04C6 11 D5 04
0497   04C9 12 40 08
0497   04CC 11 D6 04
0497   04CF 12 41 08
0497   04D2 13 BF 05
0497   04D5 D7 04
0498   04D7
0499   04D7             ;Get hex input string for address
0500   04D7                         call0(get_line)
0500   04D7
0500   04D7 11 E6 04
0500   04DA 12 40 08
0500   04DD 11 E7 04
0500   04E0 12 41 08
0500   04E3 13 65 05
0500   04E6 E8 04
0501   04E8
0502   04E8             ;Write newline
0503   04E8 11 09 07                ldm     new_line
0504   04EB 12 34 08                stm     ws_inst+1
0505   04EE 11 0A 07                ldm     new_line+1
0506   04F1 12 35 08                stm     ws_inst+2
0507   04F4                         call0(write_string)
0507   04F4
0507   04F4 11 03 05
0507   04F7 12 40 08
0507   04FA 11 04 05
0507   04FD 12 41 08
0507   0500 13 BF 05
0507   0503 05 05
0508   0505
0509   0505             ;No error checking for length of string -- must be exactly 4 hex characters
0510   0505             ;Memory address stored in address variable
0511   0505 11 80 08                ldm     buffer          ;first character
0512   0508 12 0A 08                stm     char_pair
0513   050B 11 81 08                ldm     buffer+1        ;second character
0514   050E 12 0B 08                stm     char_pair+1
0515   0511                         call1(hex_pair_to_byte)
0515   0511
0515   0511 11 20 05
0515   0514 12 43 08
0515   0517 11 21 05
0515   051A 12 44 08
0515   051D 13 2C 06
0515   0520 22 05
0516   0522 16 98 00                jpc     sm_warm         ;non-hex character detected, quit
0517   0525 12 0F 08                stm     address+1       ;high byte of address
0518   0528 11 82 08                ldm     buffer+2        ;third character
0519   052B 12 0A 08                stm     char_pair
0520   052E 11 83 08                ldm     buffer+3        ;fourth character
0521   0531 12 0B 08                stm     char_pair+1
```

```
0522   0534                          call1(hex_pair_to_byte)
0522   0534
0522   0534 11 43 05
0522   0537 12 43 08
0522   053A 11 44 05
0522   053D 12 44 08
0522   0540 13 2C 06
0522   0543 45 05
0523   0545 16 98 00                 jpc     sm_warm         ;non-hex character detected, quit
0524   0548 12 0E 08                 stm     address         ;low byte of address
0525   054B               ;Switch for return jumps
0526   054B 11 11 08                 ldm     choice
0527   054E 0F 35                    cmp     '5'
0528   0550 16 F0 02                 jpc     bl_addr_back
0529   0553 0F 34                    cmp     '4'
0530   0555 16 0C 02                 jpc     ld_addr_back
0531   0558 0F 33                    cmp     '3'
0532   055A 16 FA 01                 jpc     run_addr_back
0533   055D 0F 32                    cmp     '2'
0534   055F 16 E7 00                 jpc     d_addr_back
0535   0562 13 98 00                 jmp     sm_warm
0536   0565
0537   0565               ;Get_line subroutine, call as level 0
0538   0565               ;Gets a line from input, puts zero-terminated string in buffer
0539   0565               ;Echos characters to screen, except terminating carriage return
0540   0565               ;Address of buffer in buff_low and buff_high constants
0541   0565               ;Uses RAM variable address instruction gl_indexed_stm
0542   0565               ;length of input string returned in gl_str_len
0543   0565               ;Returns when carriage return entered
0544   0565
0545   0565 10 00         get_line:    ldi     0
0546   0567 12 07 08                   stm     gl_str_len      ;string length
0547   056A 10 80                      ldi     buff_low        ;low byte of buffer address
0548   056C 12 3A 08                   stm     gl_indexed_stm+1
0549   056F 10 08                      ldi     buff_high       ;high byte of buffer address
0550   0571 12 3B 08                   stm     gl_indexed_stm+2
0551   0574 17 03         gl_chk_loop: in      03h             ;get status
0552   0576 0C 02                      andim   02h             ;check RxRDY
0553   0578 14 74 05                   jpz     gl_chk_loop
0554   057B 17 02                      in      02h             ;get char from port
0555   057D 12 10 08                   stm     temp            ;save character
0556   0580 0A 0D                      subim   0dh             ;is it a return character?
0557   0582 14 92 05                   jpz     gl_end_of_line ;yes, replace with a zero
0558   0585 11 07 08                   ldm     gl_str_len      ;no, increment string length
0559   0588 19                         inc
0560   0589 12 07 08                   stm     gl_str_len
0561   058C 11 10 08                   ldm     temp            ;get back char
0562   058F 13 95 05                   jmp     gl_store_it     ;place character in buffer
0563   0592 12 10 08    gl_end_of_line:      stm     temp            ;place zero in temp
0564   0595 13 39 08    gl_store_it: jmp     gl_indexed_stm ;store character in buffer
0565   0598 11 10 08    gl_back:     ldm     temp            ;check if end-of-line (temp = 0)
```

```
0566   059B 14 BC 05                jpz     gl_done         ;yes, quit
0567   059E 17 03       gl_out_loop: in      03h             ;no, send char to screen
0568   05A0 0C 01                   andim   01h             ;check TxRDY
0569   05A2 14 9E 05                jpz     gl_out_loop     ;loop if not ready
0570   05A5 11 10 08                ldm     temp
0571   05A8 18 02                   out     02h             ;output character to port
0572   05AA 11 3A 08                ldm     gl_indexed_stm+1      ;increment indexing pointer
0573   05AD 19                      inc
0574   05AE 12 3A 08                stm     gl_indexed_stm+1
0575   05B1 11 3B 08                ldm     gl_indexed_stm+2
0576   05B4 09 00                   adcim   00h             ;16-bit addition
0577   05B6 12 3B 08                stm     gl_indexed_stm+2
0578   05B9 13 74 05                jmp     gl_chk_loop
0579   05BC              gl_done:   ret0
0579   05BC 13 3F 08
0580   05BF
0581   05BF              ;Write_string subroutine, call as level 0
0582   05BF              ;Writes a zero-terminated string to screen at current cursor location
0583   05BF              ;Must set up address of string to be written in ws_inst+1 and ws_inst+2
0584   05BF              write_string:
0585   05BF 17 03       ws_chk_loop: in      03h     ;get status
0586   05C1 0C 01                   andim   01h     ;check TxRDY
0587   05C3 14 BF 05                jpz     ws_chk_loop
0588   05C6 13 33 08                jmp     ws_inst ;get a character when port ready
0589   05C9 14 E0 05    ws_back:    jpz     ws_done ;quit if end-of-string
0590   05CC 18 02                   out     02h     ;output character to port
0591   05CE 11 34 08                ldm     ws_inst+1      ;indexing pointer
0592   05D1 19                      inc
0593   05D2 12 34 08                stm     ws_inst+1
0594   05D5 11 35 08                ldm     ws_inst+2
0595   05D8 09 00                   adcim   00h     ;16-bit addition
0596   05DA 12 35 08                stm     ws_inst+2
0597   05DD 13 BF 05                jmp     ws_chk_loop
0598   05E0              ws_done:   ret0
0598   05E0 13 3F 08
0599   05E3
0600   05E3              ;Subroutine hex_to word -- call as level 0
0601   05E3              ;Calls hex_pair_to_byte as level 1
0602   05E3              ;Get 16-bit word value from input string in buffer
0603   05E3              ;No error checking for length of string -- must be exactly 4 hex characters
0604   05E3              ;16-bit value placed in h2w_value
0605   05E3 11 80 08    hex_to_word: ldm     buffer         ;first character
0606   05E6 12 0A 08                stm     char_pair
0607   05E9 11 81 08                ldm     buffer+1       ;second character
0608   05EC 12 0B 08                stm     char_pair+1
0609   05EF                         call1(hex_pair_to_byte)       ;high-order byte
0609   05EF
0609   05EF 11 FE 05
0609   05F2 12 43 08
0609   05F5 11 FF 05
0609   05F8 12 44 08
```

```
0609   05FB 13 2C 06
0609   05FE 00 06
0610   0600 16 29 06               jpc     h2w_done       ;non-hex character detected, exit with carry set
0611   0603 12 0D 08               stm     h2w_value+1    ;high byte of address
0612   0606 11 82 08               ldm     buffer+2       ;third character
0613   0609 12 0A 08               stm     char_pair
0614   060C 11 83 08               ldm     buffer+3       ;fourth character
0615   060F 12 0B 08               stm     char_pair+1
0616   0612                        call1(hex_pair_to_byte)      ;low-order byte
0616   0612
0616   0612 11 21 06
0616   0615 12 43 08
0616   0618 11 22 06
0616   061B 12 44 08
0616   061E 13 2C 06
0616   0621 23 06
0617   0623 16 29 06               jpc     h2w_done       ;exit with carry set if error
0618   0626 12 0C 08               stm     h2w_value      ;low byte of address
0619   0629            h2w_done    ret0
0619   0629 13 3F 08
0620   062C
0621   062C            ;Subroutine to convert hex character pair to byte, call as level 1
0622   062C            ;Character pair in memory location char_pair, stored hi-low
0623   062C            ;Returns with byte in accumulator and carry flag clear if no error
0624   062C            ;Returns with character in accumulator and carry flag set if error
0625   062C            ;Calls char_to_nybble as level 2 subroutine
0626   062C 11 0A 08    hex_pair_to_byte: ldm      char_pair      ;high order character of pair
0627   062F 12 10 08               stm     temp           ;char_to_nybble needs char in TEMP
0628   0632                        call2(char_to_nybble)
0628   0632
0628   0632 11 41 06
0628   0635 12 46 08
0628   0638 11 42 06
0628   063B 12 47 08
0628   063E 13 D7 06
0628   0641 43 06
0629   0643 16 88 06               jpc     c2n_error
0630   0646 12 12 08               STM     byte           ;Will eventually contain the byte to load
0631   0649 12 10 08               STM     temp           ;Value to add
0632   064C 10 10                  LDI     10H            ;Multiply x 16 to shift into high-order nybble
0633   064E 12 13 08               STM     counter
0634   0651 11 13 08    MULTLOOP:    LDM     counter
0635   0654 1A                     DEC
0636   0655 14 67 06               JPZ     GET_LO         ;Have added 16 times, done
0637   0658 12 13 08               STM     counter
0638   065B 11 10 08               LDM     temp           ;Original nybble
0639   065E 00 12 08               ADD     byte           ;Add to BYTE and store
0640   0661 12 12 08               STM     byte
0641   0664 13 51 06               JMP     MULTLOOP       ;Keep adding
0642   0667 11 0B 08    GET_LO:              ldm     char_pair+1
0643   066A 12 10 08               stm     temp
```

```
0644   066D                          call2(char_to_nybble)
0644   066D
0644   066D 11 7C 06
0644   0670 12 46 08
0644   0673 11 7D 06
0644   0676 12 47 08
0644   0679 13 D7 06
0644   067C 7E 06
0645   067E 16 88 06                 jpc     c2n_error
0646   0681 00 12 08                 ADD     byte            ;Combine with hi nybble stored in BYTE
0647   0684 1C                       ccf                     ;in case addition changed it
0648   0685 13 89 06                 JMP     c2b_done        ;Done, no error
0649   0688 1B            c2n_error: scf
0650   0689               c2b_done:  ret1
0650   0689 13 42 08
0651   068C
0652   068C               ;Subroutine to convert byte to hex character pair, call as level 0
0653   068C               ;Gets byte from byte variable
0654   068C               ;Returns with character pair in memory location char_pair, stored hi-low
0655   068C               ;
0656   068C               byte_to_hex_pair:
0657   068C 11 12 08                 ldm     byte
0658   068F 0C F0                    andim   0f0h            ;dealing with high nybble
0659   0691 12 10 08                 stm     temp
0660   0694 10 00                    ldi     00h             ;prepare to shift down (divide by 16)
0661   0696 12 13 08                 stm     counter
0662   0699 11 10 08      b2h_divide: ldm    temp
0663   069C 0A 10                    subim   16
0664   069E 16 AE 06                 jpc     b2h_cont        ;continue if nybble >=0
0665   06A1 11 13 08                 ldm     counter         ;hi-nybble now in low position
0666   06A4 0F 0A                    cmp     10              ;is value <10?
0667   06A6 16 BB 06                 jpc     b2h_hi_A2F      ;yes, jump and convert to char
0668   06A9 08 30                    addim   30h             ;no, convert to char
0669   06AB 13 BD 06                 jmp     b2h_store_hi
0670   06AE 12 10 08      b2h_cont:  stm     temp
0671   06B1 11 13 08                 ldm     counter
0672   06B4 19                       inc
0673   06B5 12 13 08                 stm     counter
0674   06B8 13 99 06                 jmp     b2h_divide
0675   06BB 08 37         b2h_hi_A2F: addim  37h
0676   06BD 12 0A 08      b2h_store_hi:      stm     char_pair       ;hi-order hex char
0677   06C0 11 12 08                 ldm     byte
0678   06C3 0C 0F                    andim   0fh             ;dealing with low-order nybble
0679   06C5 0F 0A                    cmp     10              ;is value <10?
0680   06C7 16 CF 06                 jpc     b2h_lo_A2F      ;yes, jump and convert to char
0681   06CA 08 30                    addim   30h             ;no, convert to char
0682   06CC 13 D1 06                 jmp     b2h_store_lo
0683   06CF 08 37         b2h_lo_A2F: addim  37h
0684   06D1 12 0B 08      b2h_store_lo:      stm     char_pair+1     ;now char pair is in variable
0685   06D4                          ret0
0685   06D4 13 3F 08
```

```
0686   06D7
0687   06D7
0688   06D7             ;Subroutine to convert hex char to nybble, call as level 2
0689   06D7             ;Checks for validity of char, 0-9 and A-F (upper case only)
0690   06D7             ;Carry flag set on exit if error
0691   06D7             ;Carry flag clear if character valid
0692   06D7             ;Call with char in temp
0693   06D7             ;Exits with nybble in lower half of accumulator if no error
0694   06D7             ;Original character in accumulator if error
0695   06D7
0696   06D7 11 10 08    char_to_nybble:     ldm     temp           ;Get character
0697   06DA 0F 30                           cmp     30H            ;Lower limit of hex characters
0698   06DC 16 E2 06                        jpc     c2n_next       ;Char >= 30H, possibly valid
0699   06DF 13 02 07                        jmp     invalid        ;Char < 30H, invalid hex char
0700   06E2 0F 47       c2n_next:           cmp     47h            ;ASCII for "G"
0701   06E4 16 02 07                        jpc     invalid        ;Char is G or greater, invalid
0702   06E7 0F 41                           cmp     41h            ;ASCII for "A"
0703   06E9 16 F4 06                        jpc     validAF        ;Char is valid A-F
0704   06EC 0F 3A                           cmp     3Ah            ;ASCII for ":"
0705   06EE 16 02 07                        jpc     invalid        ;Char is ":" or greater, but < "A", invalid
0706   06F1 13 FC 06                        jmp     valid09        ;Char is valid 0-9
0707   06F4 0C 0F       validAF:            andim   0fh            ;Mask off high bits
0708   06F6 08 09                           addim   9              ;Adjust ASCII to binary value
0709   06F8 1C                              ccf                    ;exit no error
0710   06F9                                 ret2
0710   06F9 13 45 08
0711   06FC 0C 0F       valid09:            andim   0fh            ;Mask off high bits
0712   06FE 1C                              ccf                    ;exit no error
0713   06FF                                 ret2
0713   06FF 13 45 08
0714   0702 11 10 08    invalid:            ldm     temp           ;put char in accumulator
0715   0705 1B                              scf                    ;Set carry flag
0716   0706                                 ret2
0716   0706 13 45 08
0717   0709             ;String constants
0718   0709 0B 07       new_line            .dw     $+2
0719   070B 0D 0A 00                        .db     0dh,0ah,0
0720   070E 10 07       sm_greeting:        .dw     $+2
0721   0710 0D 0A                           .db     0dh,0ah
0722   0712 43505576696C                    .text   "CPUville 8-bit processor system monitor v.1"
0722   0718 6C6520382D6269742070726F636573736F722073797374656D206D6F6E69746F7220762E31
0723   073D 0D 0A 00                        .db     0dh,0ah,0
0724   0740 42 07       sm_prompt           .dw     $+2
0725   0742 0D 0A                           .db     0dh,0ah
0726   0744 456E74657220                    .text   "Enter number: 1=restart 2=dump 3=run 4=load 5=bload "
0726   074A 6E756D6265723A20313D726573746172742032 3D64756D7020333D72756E20343D6C6F616420353D626C6F616420
0727   0778 00                              .db     0
0728   0779 7B 07       addr_str            .dw     $+2
0729   077B 0D 0A                           .db     0dh,0ah
0730   077D 416464726573                    .text   Address (hex):
0730   0783 732028686578293A20
```

```
0731   078C 00                        .db     0
0732   078D 8F 07       bytes_str     .dw     $+2
0733   078F 0D 0A                     .db     0dh,0ah
0734   0791 427974657320              .text   Bytes to load (dec):
0734   0797 746F206C6F6164202864656329A20
0735   07A6 00                        .db     0
0736   07A7 A9 07       bl_ready_str .dw     $+2
0737   07A9 0D 0A                     .db     0dh,0ah
0738   07AB 52656164792C              .text   Ready, start transfer
0738   07B1 207374617274207472616E73666572
0739   07C0 0D 0A 00                  .db     0dh,0ah,0
0740   07C3
0741   07C3             ;The following section contains labels for RAM variables and other structures
0742   0800                           .org    0800h          ;Start of RAM
0743   0800             ;RAM Variables
0744   0800 00 00       dp_value      .dw     0000h
0745   0802 00          dp_10000s     .db     00h
0746   0803 00          dp_1000s      .db     00h
0747   0804 00          dp_100s               .db     00h
0748   0805 00          dp_10s                .db     00h
0749   0806 00          dp_1s         .db     00h
0750   0807 00          gl_str_len    .db     00h
0751   0808 00 00       bl_byte_counter       .dw     0000h
0752   080A 00 00       char_pair     .dw     0000h
0753   080C 00 00       h2w_value     .dw     0000h
0754   080E 00 00       address               .dw     0000h
0755   0810 00          temp          .db     00h
0756   0811 00          choice                .db     00h
0757   0812 00          byte          .db     00h
0758   0813 00          counter               .db     00h
0759   0814 00          line_counter .db     00h
0760   0815 00          char_count    .db     00h
0761   0816 00          byte_counter .db     00h
0762   0817 00          nybble_counter        .db     00h
0763   0818
0764   0818             ;RAM instructions with variable address (must initialize opcode when monitor is in ROM)
0765   0818             ;Jump instruction for run routine, must be in RAM
0766   0818 13 00 00    run_jump      jmp     0000h
0767   081B             ;Indexed load for load routine, must be in RAM
0768   081B 12 00 00    ld_indexed_stm        stm     0000h
0769   081E 13 D3 02                  jmp     ld_stm_back
0770   0821             ;Indexed load and store instructions for dump, must be in RAM
0771   0821 11 00 00    d_indexed_ldm         ldm     0000h
0772   0824 13 59 01                  jmp     d_ldm_back
0773   0827 12 00 00    d_indexed_stm         stm     0000h
0774   082A 13 78 01                  jmp     d_stm_back
0775   082D             ;Indexed store instruction for binary loader, must be in RAM
0776   082D 12 00 00    bl_indexed_stm:       stm     0000h
0777   0830 13 8A 04                  jmp     bl_back
0778   0833             ;Indexed load instruction for write_string, must be in RAM
0779   0833 11 00 00    ws_inst:      ldm     0000h
```

```
0780   0836 13 C9 05                jmp     ws_back
0781   0839              ;Indexed store instruction for get_line, must be in RAM
0782   0839 12 00 00     gl_indexed_stm:      stm     0000h
0783   083C 13 98 05                jmp     gl_back
0784   083F              ;Return instruction for level 0 call macros, must be in RAM
0785   083F 13 00 00     return_jump0:        jmp     0000h
0786   0842              ;Return instruction for level 1 call macros, must be in RAM
0787   0842 13 00 00     return_jump1:        jmp     0000h
0788   0845              ;Return instruction for level 2 call macros, must be in RAM
0789   0845 13 00 00     return_jump2:        jmp     0000h
0790   0848
0791   0848              return:              .set    0000h  ;assembler needs variable label set back to original value
0792   0848                           .end
0793   0848
0794   0848
0795   0848
0796   0848
0797   0848
0798   0848
0799   0848
tasm: Number of errors = 0
```