Tiny Machine-Code Monitor-Arduino Code 18th July 2018

```
/* Tiny Machine-Code Monitor - see http://www.technoblogy.com/show?283C
  David Johnson-Davies - www.technoblogy.com - 18th July 2018
   ATtiny85 @ 8 MHz (internal oscillator; BOD disabled)
*/
#include <Wire.h>
#include <avr/sleep.h>

// 20-key Matrix keypad *******************************************************
const int Matrix = A2; // PB4
const int RUN = 17;
const int Up = 18;
const int Down = 19;
const int ON = 20;
const int SmallestGap = 24;
int AnalogVals[] = {1023, 728, 693, 657, 632, 605, 567, 541, 517, 477, 439,
                   404,  371, 316, 290, 264, 220, 186, 151, 93,  0,   -100};
int Buttons[] =    {-1,   1,   2,   3,   10,  17,  4,   7,   14,  5,   8,
                   0,    6,   9,   11,  15,  12,  18,  13,  19,  20};

// Returns the keypad character or -1 if no button pressed
int ReadKeypad() {
  int val, lastval=0, count = 0;
  do {
    val = analogRead(Matrix);
    if (abs(val-lastval)<2) count++;
    else { lastval = val; count = 0; }
  } while (count < 3);
  int i = 0;
  val = val - SmallestGap/2;
  while (val < AnalogVals[i]) { i++; }
  return Buttons[i - 1];
}

// OLED I2C 128 x 32 monochrome display ********************************************
const int OLEDAddress = 0x3C;

// Initialisation sequence for OLED module
int const InitLen = 15;
const unsigned char Init[InitLen] PROGMEM = {
  0xA8, // Set multiplex
  0x1F, // for 32 rows
  0x8D, // Charge pump
  0x14,
  0x20, // Memory mode
  0x01, // Vertical addressing
  0xA1, // 0xA0/0xA1 flip horizontally
  0xC8, // 0xC0/0xC8 flip vertically
  0xDA, // Set comp ins
  0x02,
  0xD9, // Set pre charge
  0xF1,
  0xDB, // Set vcom deselect
  0x40,
  0xAF  // Display on
};

const int data = 0x40;
const int single = 0x80;
const int command = 0x00;

void InitDisplay () {
  Wire.beginTransmission(OLEDAddress);
  Wire.write(command);
  for (uint8_t c=0; c<InitLen; c++) Wire.write(pgm_read_byte(&Init[c]));
  Wire.endTransmission();
}

void DisplayOnOff (int On) {
  Wire.beginTransmission(OLEDAddress);
  Wire.write(command);
  Wire.write(0xAE + On);
  Wire.endTransmission();
}

// Character terminal ************************************************

int Line, Column;
int Scale = 1; // 2 for big characters
const char Return = 13;

// Character set for text (upper-case only) - stored in program memory
const uint8_t CharMap[64][6] PROGMEM = {
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
{ 0x00, 0x00, 0x5F, 0x00, 0x00, 0x00 },
{ 0x00, 0x07, 0x00, 0x07, 0x00, 0x00 },
{ 0x14, 0x7F, 0x14, 0x7F, 0x14, 0x00 },
{ 0x24, 0x2A, 0x7F, 0x2A, 0x12, 0x00 },
{ 0x23, 0x13, 0x08, 0x64, 0x62, 0x00 },
{ 0x36, 0x49, 0x56, 0x20, 0x50, 0x00 },
{ 0x00, 0x08, 0x07, 0x03, 0x00, 0x00 },
{ 0x00, 0x1C, 0x22, 0x41, 0x00, 0x00 },
{ 0x00, 0x41, 0x22, 0x1C, 0x00, 0x00 },
{ 0x2A, 0x1C, 0x7F, 0x1C, 0x2A, 0x00 },
{ 0x08, 0x08, 0x3E, 0x08, 0x08, 0x00 },
{ 0x00, 0x80, 0x70, 0x30, 0x00, 0x00 },
{ 0x08, 0x08, 0x08, 0x08, 0x08, 0x00 },
{ 0x00, 0x00, 0x60, 0x60, 0x00, 0x00 },
{ 0x20, 0x10, 0x08, 0x04, 0x02, 0x00 },
{ 0x3E, 0x51, 0x49, 0x45, 0x3E, 0x00 },
{ 0x00, 0x42, 0x7F, 0x40, 0x00, 0x00 },
{ 0x72, 0x49, 0x49, 0x49, 0x46, 0x00 },
{ 0x21, 0x41, 0x49, 0x4D, 0x33, 0x00 },
{ 0x18, 0x14, 0x12, 0x7F, 0x10, 0x00 },
{ 0x27, 0x45, 0x45, 0x45, 0x39, 0x00 },
{ 0x3C, 0x4A, 0x49, 0x49, 0x31, 0x00 },
{ 0x41, 0x21, 0x11, 0x09, 0x07, 0x00 },
{ 0x36, 0x49, 0x49, 0x49, 0x36, 0x00 },
{ 0x46, 0x49, 0x49, 0x29, 0x1E, 0x00 },
{ 0x00, 0x00, 0x14, 0x00, 0x00, 0x00 },
{ 0x00, 0x40, 0x34, 0x00, 0x00, 0x00 },
{ 0x00, 0x08, 0x14, 0x22, 0x41, 0x00 },
{ 0x14, 0x14, 0x14, 0x14, 0x14, 0x00 },
{ 0x00, 0x41, 0x22, 0x14, 0x08, 0x00 },
{ 0x02, 0x01, 0x59, 0x09, 0x06, 0x00 },
{ 0x3E, 0x41, 0x5D, 0x59, 0x4E, 0x00 },
{ 0x7C, 0x12, 0x11, 0x12, 0x7C, 0x00 },
{ 0x7F, 0x49, 0x49, 0x49, 0x36, 0x00 },
{ 0x3E, 0x41, 0x41, 0x41, 0x22, 0x00 },
{ 0x7F, 0x41, 0x41, 0x41, 0x3E, 0x00 },
{ 0x7F, 0x49, 0x49, 0x49, 0x41, 0x00 },
{ 0x7F, 0x09, 0x09, 0x09, 0x01, 0x00 },
{ 0x3E, 0x41, 0x41, 0x51, 0x73, 0x00 },
{ 0x7F, 0x08, 0x08, 0x08, 0x7F, 0x00 },
{ 0x00, 0x41, 0x7F, 0x41, 0x00, 0x00 },
{ 0x20, 0x40, 0x41, 0x3F, 0x01, 0x00 },
{ 0x7F, 0x08, 0x14, 0x22, 0x41, 0x00 },
{ 0x7F, 0x40, 0x40, 0x40, 0x40, 0x00 },
{ 0x7F, 0x02, 0x1C, 0x02, 0x7F, 0x00 },
{ 0x7F, 0x04, 0x08, 0x10, 0x7F, 0x00 },
{ 0x3E, 0x41, 0x41, 0x41, 0x3E, 0x00 },
{ 0x7F, 0x09, 0x09, 0x09, 0x06, 0x00 },
{ 0x3E, 0x41, 0x51, 0x21, 0x5E, 0x00 },
{ 0x7F, 0x09, 0x19, 0x29, 0x46, 0x00 },
{ 0x26, 0x49, 0x49, 0x49, 0x32, 0x00 },
{ 0x03, 0x01, 0x7F, 0x01, 0x03, 0x00 },
{ 0x3F, 0x40, 0x40, 0x40, 0x3F, 0x00 },
{ 0x1F, 0x20, 0x40, 0x20, 0x1F, 0x00 },
{ 0x3F, 0x40, 0x38, 0x40, 0x3F, 0x00 },
{ 0x63, 0x14, 0x08, 0x14, 0x63, 0x00 },
{ 0x03, 0x04, 0x78, 0x04, 0x03, 0x00 },
{ 0x61, 0x59, 0x49, 0x4D, 0x43, 0x00 },
{ 0x00, 0x7F, 0x41, 0x41, 0x41, 0x00 },
{ 0x02, 0x04, 0x08, 0x10, 0x20, 0x00 },
{ 0x00, 0x41, 0x41, 0x41, 0x7F, 0x00 },
{ 0x04, 0x02, 0x01, 0x02, 0x04, 0x00 },
{ 0x40, 0x40, 0x40, 0x40, 0x40, 0x00 },
};

void ClearDisplay () {
  Wire.beginTransmission(OLEDAddress);
  Wire.write(command);
  // Set column address range
  Wire.write(0x21); Wire.write(0); Wire.write(127);
  // Set page address range
  Wire.write(0x22); Wire.write(0); Wire.write(3);
  Wire.endTransmission();
  // Write the data in 26 20-byte transmissions
  for (int i = 0 ; i < 26; i++) {
    Wire.beginTransmission(OLEDAddress);
    Wire.write(data);
    for (int i = 0 ; i < 20; i++) Wire.write(0);
    Wire.endTransmission();
  }
}

// Converts bit pattern abcdefgh into aabbccddeeffgghh
int Stretch (int x) {
  x = (x & 0xF0)<<4 | (x & 0x0F);
  x = (x<<2 | x) & 0x3333;
  x = (x<<1 | x) & 0x5555;
  return x | x<<1;
}

// Plots a character
void Pchar(int c) {
  Wire.beginTransmission(OLEDAddress);
  Wire.write(command);
  // Set column address range
  Wire.write(0x21); Wire.write(Column*6); Wire.write(Column*6 + Scale*6 - 1);
  // Set page address range
  Wire.write(0x22); Wire.write(Line); Wire.write(Line + Scale - 1);
  Wire.endTransmission();
  Wire.beginTransmission(OLEDAddress);
  Wire.write(data);
  for (uint8_t col = 0 ; col < 6; col++) {
    int bits = pgm_read_byte(&CharMap[c-32][col]);
    if (Scale == 1) Wire.write(bits);
    else {
      bits = Stretch(bits);
      for (int i=2; i--;) { Wire.write(bits); Wire.write(bits>>8); }
    }
  }
  Wire.endTransmission();
  Column = Column + Scale;
}

// Print text at line, column
void Print (PGM_P s) {
  int p = (int)s;
  while (1) {
    char c = pgm_read_byte(p++);
    if (c == 0) return;
    Pchar(c);
  }
}

char Hex (uint8_t byte) { return (byte < 10) ? byte+'0' : byte-10+'A'; }

void Phex (uint8_t byte) { Pchar(Hex(byte)); }

// Print a hex byte
void Pbyte (uint8_t byte) {
  Phex(byte>>4); Phex(byte & 0xF);
}

// Display a 4-digit decimal number
void PlotNumber (int num) {
  for (long d=1000; d>0; d=d/10) {
    Pchar(num/d % 10 + '0');
  }
}

// MINIL Interpreter **********************************************
const int MemorySize = 64;
const int StackSize = 72;

int stk[StackSize];
uint8_t mem[MemorySize];
int reg[8];
int pc = 0, sp = 0;
boolean err = false;

void error (PGM_P s) {
  Line =  0; Column = 0;
  Print(s);
  err = true;
  delay(1000);
}
```

```
const int Timeout = 30;                  // Sleep after this many secs

// Wait for keypress and go to sleep after Timeout
int WaitForKey () {
  int key;
  unsigned long Start = millis();
  do {
    key = ReadKeypad();
    if (millis() - Start > Timeout*1000) {
      DisplayOnOff(0);                  // Blank display
      uint8_t temp = TIMSK;
      TIMSK = 0;                        // Disable timer interrupt(s)
      GIMSK = 1<<PCIE;                  // Enable pin-change interrupt
      PCMSK = 1<<PCINT4;                // on PB4
      ADCSRA &= ~(1<<ADEN);             // Disable ADC to save power
      sleep_enable();
      sleep_cpu();
      GIMSK = 0;                        // Turn off interrupt
      TIMSK = temp;                     // Re-enable timer interrupt(s)
      DisplayOnOff(1);                  // Turn on display
      ADCSRA |= 1<<ADEN;                // Re-enable ADC
      Start = millis();
    }
  } while (key == -1);
  return key;
}

void WaitForRelease () {
  int key;
  do key = ReadKeypad(); while (key != -1);
}

boolean enter (uint8_t regno) {
  boolean edit = false, done = false;
  int key;
  Line = 0; Column = 0;
  Print(PSTR("R")); Phex(regno); Print(PSTR(" = "));
  do {
    Column = 10;
    PlotNumber(reg[regno]); Pchar(' ');
    key = WaitForKey();
    if (key >= 0 && key <= 9) {
      if (!edit) {
        reg[regno] = key;
        edit = true;
      } else {
        reg[regno] = (reg[regno] % 1000) *10 + key;
      }
    } else if (key == RUN || key == ON) done = true;
    WaitForRelease();
  } while (!done);
  ClearDisplay();
  return (key == ON);
}

boolean Label (int pc) {
  if (pc == 0) return true;
  for (int x=0; x<MemorySize; x++) {
    uint8_t m = mem[x];
    if ((m >= 0x80) && ((m & 0x1F) == pc)) return true;
  }
  return false;
}

void Disassemble (int pc) {
  uint8_t high, low, jump;
  Column = 0;
  //
  // Address and instruction
  Pbyte(pc); Pchar(' ');
  Pbyte(mem[pc]); Pchar(' ');
  //
  // Optional label
  if (Label(pc)) { Pchar('L'); Pbyte(pc); Pchar(':'); Pchar(' '); }
  else Print(PSTR("     "));
  //
  // Assembler code
  uint8_t byte = mem[pc];
  high = byte>>4 & 0xf;
  low = byte & 0xf;
  jump = byte & 0x1f;
  if (high == 0 && low == 0) {
    Print(PSTR("BRK     "));
  } else if (high == 1 && low == 1) {
    Print(PSTR("NOP     "));
  } else if (high == 6 && low == 6) {
    Print(PSTR("TOG     "));
  } else if (high == 7 && low == 7) {
    Print(PSTR("RTS     "));
  } else if (high < 8 && low < 8) {
    Print(PSTR("MOV R")); Phex(high); Print(PSTR(",R")); Phex(low);
  } else if (high < 8) {
    if (low == 0x8) Print(PSTR("PSH R"));
    else if (low == 0x9) Print(PSTR("POP R"));
    else if (low == 0xA) Print(PSTR("ADD R"));
    else if (low == 0xB) Print(PSTR("SUB R"));
    else if (low == 0xC) Print(PSTR("CPY #"));
    else if (low == 0xD) Print(PSTR("DEC R"));
    else if (low == 0xE) Print(PSTR("ENT R"));
    else if (low == 0xF) Print(PSTR("??? R"));
    Phex(high); Print(PSTR("  "));
  } else if (high >= 0x8) {
    if (high <= 0x9) Print(PSTR("JZ "));
    else if (high <= 0xB) Print(PSTR("JNZ"));
    else if (high <= 0xD) Print(PSTR("JC "));
    else Print(PSTR("JSR"));
    Print(PSTR(" ")); Pchar('L'); Pbyte(jump); Print(PSTR(" "));
  }
}

void Run () {
  int pc = 0, sp = 0;
  boolean zero = false, carry = false;
  uint8_t byte, high, low, jump;
  err = false;
  for (int r=0; r<8; r++) reg[r]=0;
  do {
    byte = mem[pc++];
    high = byte>>4 & 0xf;
    low = byte & 0xf;
    jump = byte & 0x1f;
    if (high == 0 && low == 0) {
      error(PSTR("BREAK"));
    } else if (high == 6 && low == 6) {
      digitalWrite(1, !digitalRead(1));
    } else if (high == 7 && low == 7) {
      if (sp == 0) error(PSTR("STACK <"));
      pc = stk[--sp];
    } else if (high < 8 && low < 8) {
      reg[high] = reg[low];
    } else if (high < 8) {
      // Single register operations
      if (low == 0x8) {
        if (sp >= StackSize-1) error(PSTR("STACK >"));
        stk[sp++] = reg[high];
      } else if (low == 0x9) {
        if (sp == 0) error(PSTR("STACK <"));
        reg[high] = stk[--sp];
      } else if (low == 0xA) {
        carry = (reg[0] + reg[high]) > 9999;
        reg[0] = reg[0] + reg[high];
        if (carry) reg[0] = reg[0] - 10000;
        zero = reg[0] == 0;
      } else if (low == 0xB) {
        carry = reg[high] > reg[0];
        reg[0] = reg[0] - reg[high];
        if (carry) reg[0] = reg[0] + 10000;
        zero = reg[0] == 0;
      } else if (low == 0xC) {
        reg[0] = high;
      } else if (low == 0xD) {
        carry = reg[high] == 0;
        reg[high]--;
        if (carry) reg[high] = 9999;
        zero = reg[high] == 0;
      } else if (low == 0xE) {
        err = enter(high);
      }
    } else if (high >= 0x8) {
      // Jumps
      if (high <= 0x9) { if (zero) pc = jump;}
      else if (high <= 0xB) { if (!zero) pc = jump;}
      else if (high <= 0xD) { if (carry) pc = jump;}
      else {
        if (sp >= StackSize-1) error(PSTR("STACK >"));
        stk[sp++] = pc;
        pc = jump;
      }
    }
  } while (err == false && digitalRead(4) == 1);
  digitalWrite(1, LOW);
}

// Display screen with pc on bottom line
void DisplayScreen (int pc) {
  pc = pc - 3;
  for (int l=0; l<4; l++) {
    Line = l; Column = 0;
    if (pc >= 0) Disassemble(pc);
    else Print(PSTR("          "));
    pc++;
  }
}

// Setup ********************************************

void setup() {
  pinMode(1,OUTPUT);
  Wire.begin();
  set_sleep_mode(SLEEP_MODE_PWR_DOWN);
  InitDisplay();
  ClearDisplay();
  Scale = 2;
  Line = 0; Column = 0;
  Print(PSTR("TINY MINIL"));
  delay(1000);
}

// PCINT0 interrupt wakes from sleep
ISR(PCINT0_vect) { }

void loop() {
  int key, key0 = -1;
  boolean done = false;
  //
  // Entry mode
  ClearDisplay();
  Scale = 1;
  pc = 0;
  DisplayScreen(pc);
  do {
    key = WaitForKey();
    // Decode key
    if (key == Up && key0 == -1 && pc > 0) {
      pc--;
      DisplayScreen(pc);
    } else if (key == Down && key0 == -1) {
      pc++;
      DisplayScreen(pc);
    } else if (key >= 0 && key <= 15) {
      if (key0 == -1) {
        Column = 3; Phex(key); Pchar('_');
        key0 = key;
      } else {
        Column = 4; Phex(key);
        mem[pc] = key0<<4 | key;
        Disassemble(pc);
        key0 = -1;
      }
    } else if (key == RUN) done = true;
    // Wait for key release
    WaitForRelease();
  } while (!done);
  //
  // Run mode
  ClearDisplay();
  Scale = 2;
  Run();
}
```