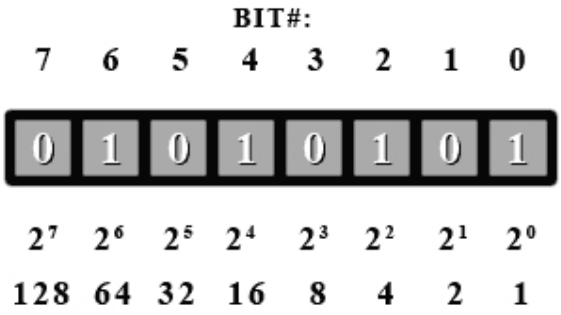
Utilizarea limbajului de asamblare în Arduino IDE

Ce este ”Inline Assembler”?



Facilitatea de ”Inline Assembler” permite inserarea de cod în limbaj de asamblare în cadrul programelor de C/C++ compilate cu ajutorul GCC (compilatorul utilizat de mediul Arduino IDE). Utilizarea de cod în limbaj de asamblare permite optimizarea unor porțiuni de cod și obținerea unor programe mai mici ca dimensiune (în format binar). Pentru mai multe informații se recomandă consultarea materialului „Inline Assembler Cookbook” [1].

Inserarea de cod în limbaj de asamblare se face utilizând directiva asm (sau ___asm___) direct în program. De exemplu (instrucțiunea NOP în limbaj de asamblare nu are nici un efect [2]):

```
asm ( "nop \n");
```

Bineînțeles, în cadrul secțiunii de cod în limbaj de asamblare nu vom beneficia de aceleași avantaje și înlesniri ca într-un program obișnuit în limbaj de nivel înalt. Pentru a înțelege mai bine vom reface exemplul clasic din Arduino IDE – programul Blink:

```
void setup() {
  pinMode(13, OUTPUT);
}
void loop() {
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

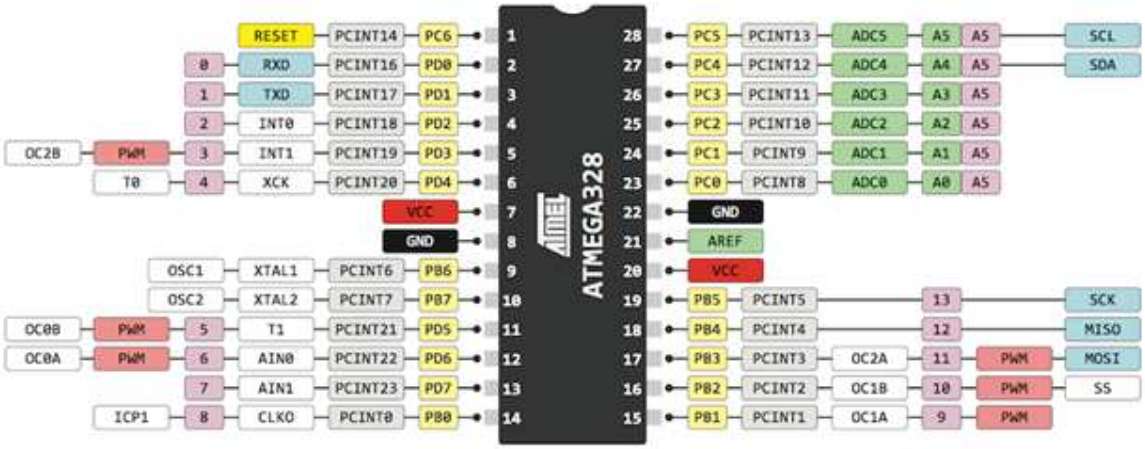
O variantă a acestui program utilizând directiva asm este:

```
void setup() {
 asm("sbi  0x04, 0x5 \n");
}
void loop() {
 asm("cbi  0x05, 0x5 \n");
 delay(1000);
 asm("sbi  0x05, 0x5 \n");
 delay(1000);
}
```

După cum se poate observa instrucțiunile pinMode și digitalWrite, specifice mediului Arduino IDE, au fost înlocuite cu instrucțiuni în limbaj de asamblare: sbi și cbi ce permit setarea sau ștergerea unui bit de la o anumită adresă din memorie. Mai mult decât atâta, nu am mai folosit referința la pinul plăcii Arduino așa cum suntem obișnuiți (pinul 13) ci adrese de memorie la care se află registrele interne de configurare ale pinului (registrul de sens DDRB – adresa 0x04 și registrul de ieșire PORTB – adresa 0x05, în ambele registre am manipulat bitul 5 corespondent pinului PB5 adică pinul 13 al plăcii Arduino). Comparați memoria program ocupată de exemplul original și cel care utilizează directiva asm.

Registre interne și echivalarea pinilor între mediul Arduino și arhitectura microcontrolerului ATmega328P

Pentru a ușura lucrul cu pinii I/O mediul Arduino IDE are propria modalitate de identificare a acestora (D0-D13, A0-A5) dar în realitate aceștia sunt organizați în trei porturi a câte 8 pini (PB0-PB7, PC0-PC7, PD0-PD7), echivalența între cele două organizări este reprezentată în diagrama următoare (nu toți pinii sunt prezenți la varianta THT a circuitului ATmega328P):

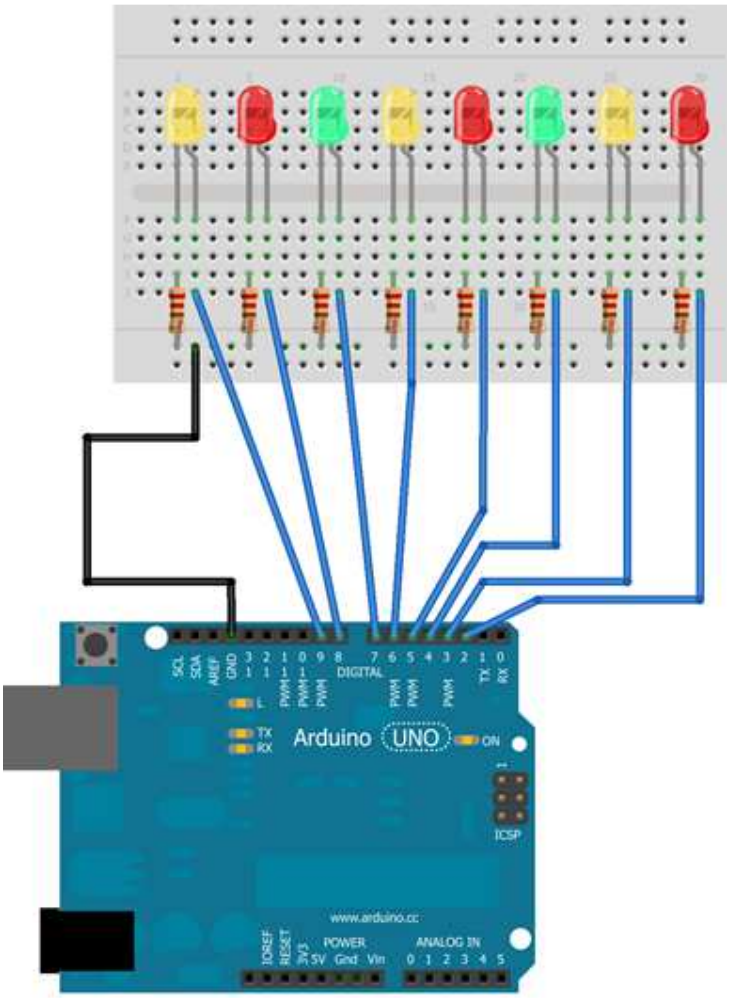


Pentru a putea manipula pinii microcontrolerului (la nivel de limbaj de asamblare) este nevoie să cunoaștem adresele registrelor DDRx (registrul de sens) și PORTx (registrul de ieșire) al portului din care face parte pinul. Pentru mai multe informații despre organizarea internă a registrelor interne este utilă consultarea manualului circuitului ATmega328P [3].

0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
0x08 (0x28)	PORTC	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
0x07 (0x27)	DDRC	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
0x06 (0x26)	PINC	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0

Exemplu de program: joc de lumini

Presupunem următoarea schemă de interconectare a 8 leduri cu placa de dezvoltare Arduino Uno în mod individual – fiecare led este comandat în mod direct de câte un pin al plăcii de dezvoltare (led 1 – pin 2, led 2 – pin 3…. led 8 – pin 9):



Funcționalitatea sistemului va consta în realizarea a două jocuri de lumini. Secțiunea setup a programului va trebui să configureze toți cei opt pini utilizați ca fiind pini de ieșire. Varianta inițială ce utilizează instrucțiunea pinMode este:

```
void setup() {
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
```

```
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
  Serial.begin(9600);
}
```

Prima variantă propusă utilizează instrucțiunea sbi ca și în exemplul precedent (adresa 0x0a este adresa registrului DDRD iar adresa 0x04 adresa registrului DDRB):

```
 asm (
  "sbi 0x0a, 2 \n"
  "sbi 0x0a, 3 \n"
  "sbi 0x0a, 4 \n"
  "sbi 0x0a, 5 \n"
  "sbi 0x0a, 6 \n"
  "sbi 0x0a, 7 \n"
  "sbi 0x04, 0 \n"
  "sbi 0x04, 1 \n"
 );
```

O variantă mai scurtă este configurarea biților din cele două registre (DDRD și DDRB) simultan utilizând registrul de uz general R26 ca intermediar pentru transmiterea valorii către cele două registre:

```
asm (
    "ldi r26, 0b11111100 \n"
    "out 0x0a, r26 \n"
    "ldi r26, 0b00000011 \n"
    "out 0x04, r26 \n"
    : : : "r26"
    );
```

Utilizarea unui registru de uz general trebuie semnalizată compilatorului pentru a nu apărea suprapuneri în utilizarea registrelor – ultima linie din cod, a se vedea și materialul „Arduino Inline Assembly Tutorial #3 (Clobbers)” [4].

Prima variantă de joc de lumini va aprinde alternativ la un interval de 1 secundă ledurile de rang impar și ledurile de rang par (led 1, led 3, led 5, led 7 – led 2, led 4, led 6, led 8). Secțiunea loop (utilizând cod Arduino) este:

```
void loop() {
  for (int i=2; i<10; i++) {
      if ((i%2)==0) digitalWrite(i,HIGH);
      else digitalWrite(i,LOW);
   }
   delay(1000);
   for (int i=2; i<10; i++) {
      if ((i%2)==0) digitalWrite(i,LOW);
      else digitalWrite(i,HIGH);
   }
   delay(1000);
}
```

Transpunerea în limbaj de asamblare este:

```
void loop() {
asm (
      "sbi 0x0b,2 \n"
      "cbi 0x0b,3 \n"
      "sbi 0x0b,4 \n"
      "cbi 0x0b,5 \n"
      "sbi 0x0b,6 \n"
      "cbi 0x0b,7 \n"
      "sbi 0x05,0 \n"
      "cbi 0x05,1 \n"
```

```
    "ldi r25, 0x7F \n"
    "wait1: ldi r26, 0xFF \n"
    "wait2: ldi r27, 0xFF \n"
    "wait3: dec r27 \n"
    "nop \n"
    "brne wait3 \n"
    "dec r26 \n"
    "brne wait2 \n"
    "dec r25 \n"
    "brne wait1 \n"
    "cbi 0x0b,2 \n"
    "sbi 0x0b,3 \n"
    "cbi 0x0b,4 \n"
    "sbi 0x0b,5 \n"
    "cbi 0x0b,6 \n"
    "sbi 0x0b,7 \n"
    "cbi 0x05,0 \n"
    "sbi 0x05,1 \n"
    "ldi r25, 0x7F \n"
    "wait1b: ldi r26, 0xFF \n"
    "wait2b: ldi r27, 0xFF \n"
    "wait3b: dec r27 \n"
    "nop \n"
    "brne wait3b \n"
    "dec r26 \n"
    "brne wait2b \n"
    "dec r25 \n"
    "brne wait1b \n"
    : : : "r25" , "r26", "r27"
);
```

Cea de a doua variantă de joc de lumini va aprinde unul câte unul (pornind de la led-ul

1 până la led-ul 8) toate led-urile și apoi le va stinge în mod similar (în ordine inversă). Operația de aprindere sau stingere a unui led se va efectua la un interval de 500 milisecunde. Secțiunea loop (utilizând cod Arduino) este:

```
void loop() {
   for (int i=2;i<10;i++) {
     digitalWrite(i,HIGH);
     delay(500); }
   for (int i=9;i>1;i--) {
     digitalWrite(i,LOW);
     delay(500);
   }
  }
```

Transpunerea în limbaj de asamblare este:

```
void loop() {
asm (
      "start: sbi 0x0b,2 \n"
      "rcall wait \n"
      "sbi 0x0b,3 \n"
      "rcall wait \n"
      "sbi 0x0b,4 \n"
      "rcall wait \n"
      "sbi 0x0b,5 \n"
      "rcall wait \n"
      "sbi 0x0b,6 \n"
      "rcall wait \n"
      "sbi 0x0b,7 \n"
      "rcall wait \n"
      "sbi 0x05,0 \n"
      "rcall wait \n"
      "sbi 0x05,1 \n"
```

```
        "rcall wait \n"
        "cbi 0x0b,2 \n"
        "rcall wait \n"
        "cbi 0x0b,3 \n"
        "rcall wait \n"
        "cbi 0x0b,4 \n"
        "rcall wait \n"
        "cbi 0x0b,5 \n"
        "rcall wait \n"
        "cbi 0x0b,6 \n"
        "rcall wait \n"
        "cbi 0x0b,7 \n"
        "rcall wait \n"
        "cbi 0x05,0 \n"
        "rcall wait \n"
        "cbi 0x05,1 \n"
        "rcall wait \n"
        "rjmp start \n"
        "wait: ldi r25, 0x3F \n"
        "wait12: ldi r26, 0xFF \n"
        "wait22: ldi r27, 0xFF \n"
        "wait32: dec r27 \n"
        "nop \n"
        "brne wait32 \n"
        "dec r26 \n"
        "brne wait22 \n"
        "dec r25 \n"
        "brne wait12 \n"
        "ret \n"
        : : : "r25" , "r26", "r27"
    );
}
```

Ambele jocuri de lumini utilizează instrucțiunile sbi și cbi pentru manipularea pinilor I/0 (pentru o listă completă a instrucțiunilor în limbaj de asamblare se recomandă parcurgerea materialului „AVR Instruction Set Manual” [5]). Funcția de temporizare delay din mediul Arduino IDE este înlocuită cu o succesiune de bucle ce realizează o întârziere aproximativă (în a doua variantă această succesiune este implementată ca o subrutină).

Referințe on-line

[1] AVR Libc Reference Manual Inline Assembler Cookbook

http://www.microchip.com/webdoc/avrlibcreferencemanual/inline_asm.html

[2] NOP

https://en.wikipedia.org/wiki/NOP

[3] ATmega328/P DATASHEET COMPLETE

http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf

[4] Arduino Inline Assembly Tutorial #3 (Clobbers)

https://ucexperiment.wordpress.com/2016/03/08/arduino-inline-assembly-tutorial-3/

[5] AVR Instruction Set Manual

http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-0856-AVR-Instruction-Set-Manual.pdf