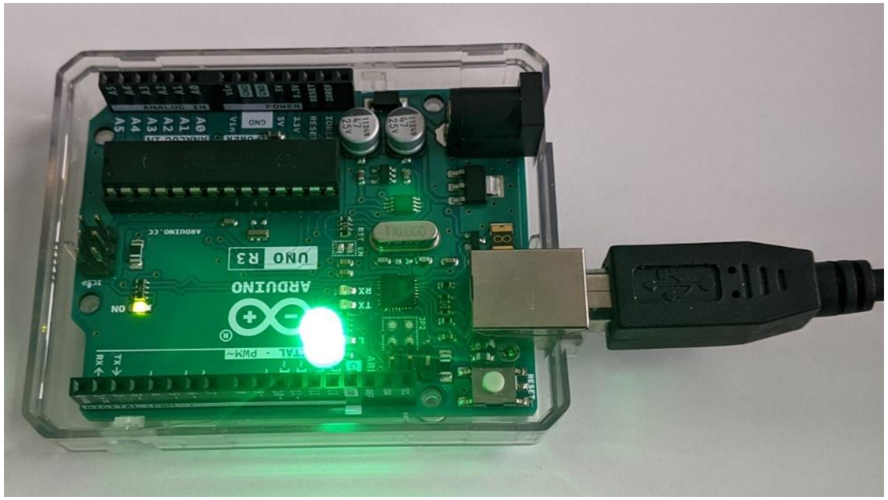
Machine Architecture - Lecture 8



Ioannis Ivrissimtzis

ioannis.ivrissimtzis@durham.ac.uk

Programming in AVR assembly

Slide material acknowledgements: Magnus Bordewich

Programming in AVR assembly

Through a simple example we will study some basic aspects of AVR assembly programming :

- directives
- I/O
- the status register
- arithmetic and logical operations
- branching

Our program will wait for a signal (e.g. the press of a button) and then a LED light will blink three times. The code was uploaded in Ultra. It is a simplified version of the example in

https://akuzechie.blogspot.com/2021/10/assembly-programming-using-arduino-uno.html

The C++ shell

An easy way to program in AVR assembly through the Arduino IDE, is to create a C++ shell file, and a separate linked file for the assembly code.

In the setup() function we will define the program's inputs and outputs.

The loop() function repeats for ever, which usually is the desirable behaviour in an embedded system.

```
//----------------------------------
// C Code: LED blinking
//----------------------------------
extern "C"
{
  void start();
  void btnLED();
}
//---------------------
void setup()
{
  start();
}
//---------------------
void loop()
{
  btnLED();
}
```

Directives

The assembly code contains some statements called directives.

Directives do not correspond to assembly instructions, but the assembler will take them into account when compiling the assembly code to machine language.

The use of directives, for example for defining a constant, makes easier to develop and maintain assembly code.

In AVR assembly a directive is precented by a dot (.).

Directives

The equ directive assigns a value to a label, and the label can then be used in later expressions.

A label assigned a value by the equ directive is a constant, it cannot be redefined. We define a constant named delayVal with value 10000 by:

```
.equ  delayVal, 10000
```

The directives .global start and .global btnLED declare the functions start and btnLED as global.

This information will be used by the linker when it puts various pieces of code from several source files together.

Declaring input output pins

```
start:
    SBI    DDRB, 4          ;set PB4     (declare pin PB4 as output pin (o/p) - LED)
    CBI    DDRD, 2          ;clear PD2  (declare pin PD2 as input pin (i/p) - e.g. a button)
    RET                     ;return to setup()
```

First, we choose a pin for output (to light the LED light), and one pin for input (to receive the signal for the light to start blinking). Here we chose pins PB4 and PD2, respectively.

The same example was used in lecture 7.

Reading values from the input pin

```
btnLED:
    SBIC  PIND, 2           ;skip next statement if button not pressed
    RJMP  blink             ;jump to label blink
    RJMP  btnLED            ;return to label btnLED
```

The status of the input pin PD4 is recorded on register PIND (lecture 7).

The SBIC instruction (**s**kip if **b**it in I/O register **i**s **c**leared) checks the corresponding bit in PIND and if it is 0 (cleared) skips one instruction.

So, if the bit (PIND,2) has value 0 (no incoming electric signal from pin PD2), we skip one instruction and go to the jump with RJMP btnLED, that is, to the beginning of the function to check again the value of that bit.

Otherwise, we jump with RJMP btnLED to the function that will blink the LED three times.

Controlling the output pin

LDI initialises the counter R21 to 3, so it would blink three times.

Pin PB4 has already been declared as output pin (bit DDRB,4 set). Its behaviour is controlled by the bit 4 of the corresponding I/O register PORTB.

When the value of bit PORTB,4 is 1 (set), the microcontroller sends electric signal through pin PB4 and the LED is on. When it is 0 (cleared) no electric signal is sent, and the light is off.

```
blink:
    LDI   R21, 3          ;initialise register R21

blinkOnce:
    SBI   PORTB, 4        ;turn ON LED
    RCALL myDelay         ;call subroutine myDelay
    CBI   PORTB, 4        ;turn OFF LED
    RCALL myDelay         ;call subroutine myDelay
    SUBI  R21, 1          ;decrement counter by 1
    BRNE  blinkOnce       ;loop if counter not zero
    RJMP  btnLED          ;return to label btnLED
```

Controlling the output pin

The value of the bit PORTB,4 is controlled (set or cleared) with the instructions SBI and CBI, which we have seen already.

In between, the program calls the function myDelay, which does some computations (countdown) just to waste some time. Otherwise, the blinking would be so rapid that our eyes wouldn't be able to notice it.

```
blink:
    LDI   R21, 3           ;initialise register R21

blinkOnce:
    SBI   PORTB, 4         ;turn ON LED
    RCALL myDelay          ;call subroutine myDelay
    CBI   PORTB, 4         ;turn OFF LED
    RCALL myDelay          ;call subroutine myDelay
    SUBI  R21, 1           ;decrement counter by 1
    BRNE  blinkOnce        ;loop if counter not zero
    RJMP  btnLED           ;return to label btnLED
```

Decrementing the counter

The counter is decremented with the SUBI instruction (subtract immediate).

Apart from subtracting the immediate 1 from register R21, SUBI will also check the result of this operation and update certain bits of the status register, see lecture 7.

In particular, if the result of the subtraction is 0, the value of the Z bit of the status register will become 1 (set). Otherwise, it will become 0 (cleared).

```
blink:
    LDI   R21, 3           ;initialise register R21

blinkOnce:
    SBI   PORTB, 4         ;turn ON LED
    RCALL myDelay          ;call subroutine myDelay
    CBI   PORTB, 4         ;turn OFF LED
    RCALL myDelay          ;call subroutine myDelay
    SUBI  R21, 1           ;decrement counter by 1
    BRNE  blinkOnce        ;loop if counter not zero
    RJMP  btnLED           ;return to label btnLED
```

SUBI instruction

Subtract Immediate

Operation: Rd ← Rd - K

Syntax: SUBI Rd, K

Operands: $16 \leq d \leq 31$, $0 \leq K \leq 255$

Machine language code:

0101	KKKK	dddd	KKKK

The Z bit of the status register is set (value 1) when the result Rd – K is equal to 0; it is cleared (value 0) otherwise.

I	T	H	S	V	N	Z	C	bit name
								value

Conditional branching

The BRNE instruction (Branch if Not Equal), branches to blinkOnce at the top of the loop, if and only if the Z bit of the status counter is 1.

Despite its name, BRNE does not compare numbers. It only checks the Z bit in the status register. The idea is that the value of the Z bit represents the result of a comparison done by the previous instruction, here SUBI.

```
blink:
    LDI   R21, 3            ;initialise register R21

blinkOnce:
    SBI   PORTB, 4          ;turn ON LED
    RCALL myDelay           ;call subroutine myDelay
    CBI   PORTB, 4          ;turn OFF LED
    RCALL myDelay           ;call subroutine myDelay
    SUBI  R21, 1            ;decrement counter by 1
    BRNE  blinkOnce         ;loop if counter not zero
    RJMP  btnLED            ;return to label btnLED
```

Finally, the RJMP, at the exit of the blink loop, branches back to the btnLED, that is, the microcontroller waits again for an input signal.

The myDelay function

```
.equ  delayVal, 10000          ;equate delayVal with initial count value

myDelay:
   LDI   R20, 100              ;initial count value for outer loop
outerLoop:
   LDI   R30, lo8(delayVal)    ;low byte of delayVal in R30
   LDI   R31, hi8(delayVal)    ;high byte of delayVal in R31
innerLoop:
   SBIW  R30, 1                ;subtract 1 from 16-bit value in R31, R30
   BRNE  innerLoop             ;jump if countVal not equal to 0

   SUBI  R20, 1                ;subtract 1 from R20
   BRNE  outerLoop             ;jump if R20 not equal to 0
   RET
```

Two nested loops counting down from 10000 and 100, respectively.

The SUBI instruction and the similar SBIW (subtract immediate from word) will be called 10,000 x 100 = 1,000,000 times in total.

The myDelay function

```
.equ  delayVal, 10000          ;equate delayVal with initial count value

myDelay:
   LDI   R20, 100              ;initial count value for outer loop
outerLoop:
   LDI   R30, lo8(delayVal)    ;low byte of delayVal in R30
   LDI   R31, hi8(delayVal)    ;high byte of delayVal in R31
innerLoop:
   SBIW  R30, 1                ;subtract 1 from 16-bit value in R31, R30
   BRNE  innerLoop             ;jump if countVal not equal to 0

   SUBI  R20, 1                ;subtract 1 from R20
   BRNE  outerLoop             ;jump if R20 not equal to 0
   RET
```

Notice the split of the iterations as 10000x100 rather than 1000^2.

This way the outerLoop counter is stored in a word (pair of registers R30 and R31), while the counter of innerLoop in a byte, in register R20.

Assembly programming affords this type of optimisations.

Exercise

Can you compute (at least approximately) the duration of the delay introduced by the myDelay function?

Hint: First, find how many times each instruction will be executed.

Then, find how many clock cycles will be required using the following table:

LDI	1 cycle
SUBI	1 cycle
SBIW	2 cycles
BRNE	1 cycle if condition is false 2 cycles if condition is true
RET	4 cycles

Finally, compute the timing knowing that ATMega328 runs at 16MHz.

Next …

… addressing modes

and the

instruction set