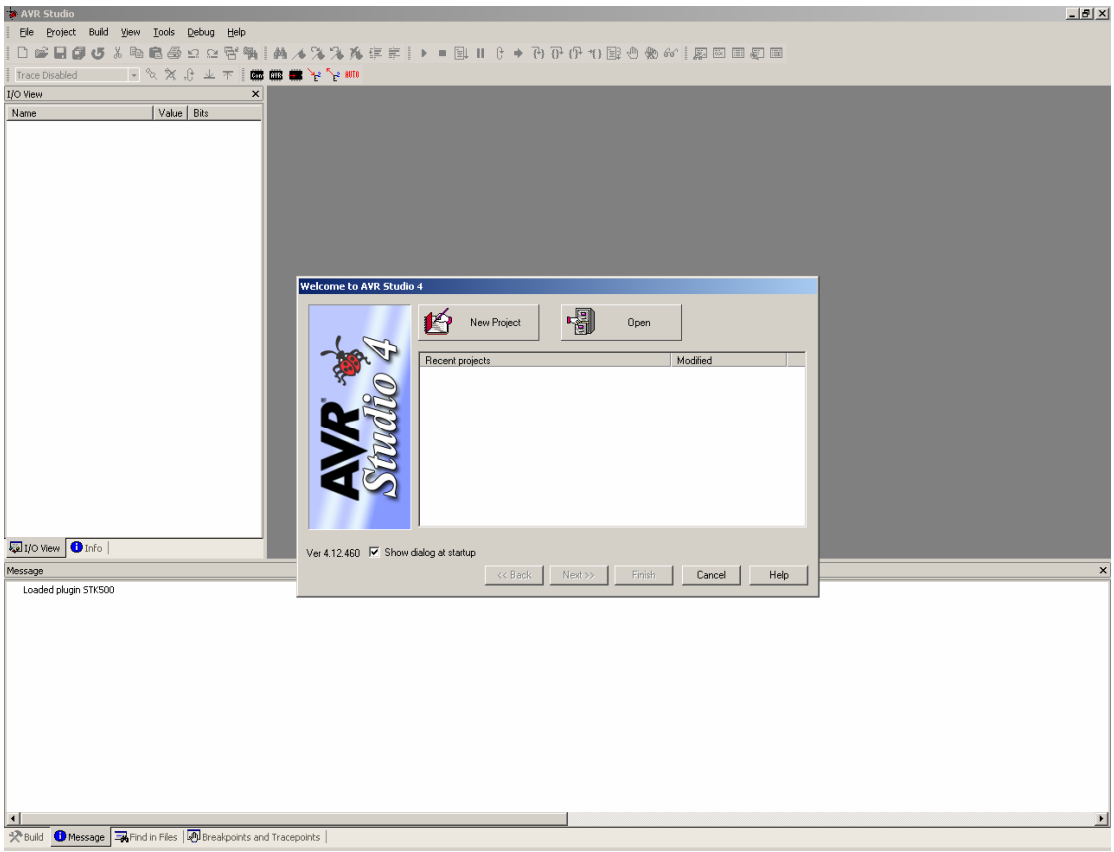
How to use AVR Studio for Assembler Programming

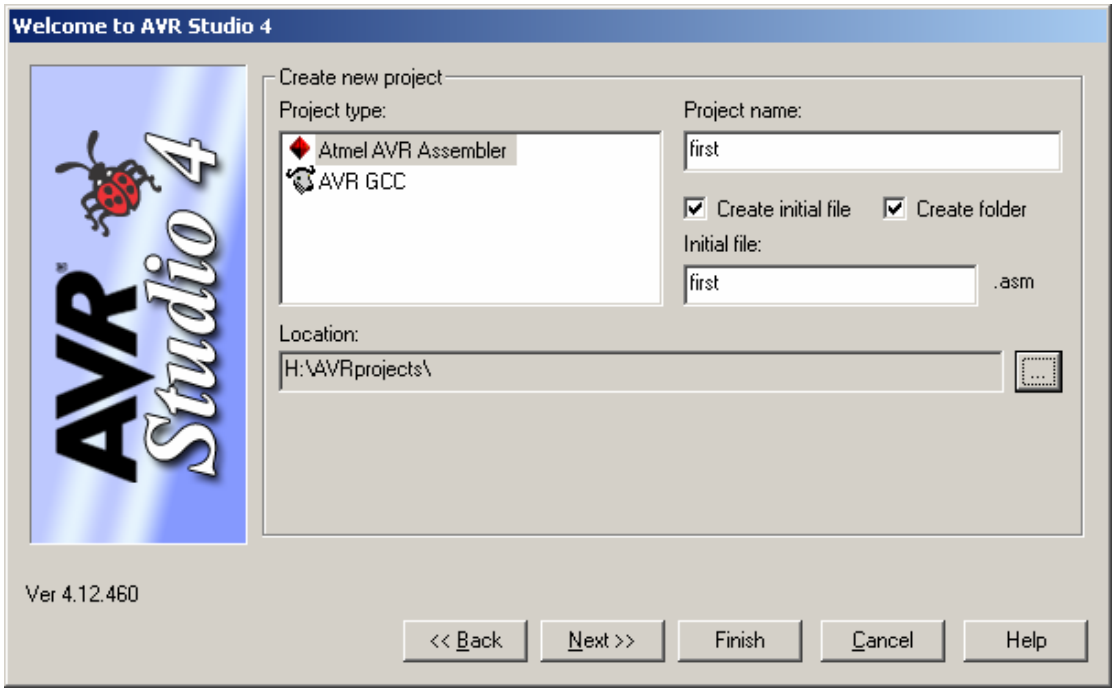
Creating your first assembler AVR project

Run the AVRStudio program by selecting **Start\Programs\Atmel AVR Tools\AVR Studio**. You should see a screen like this:

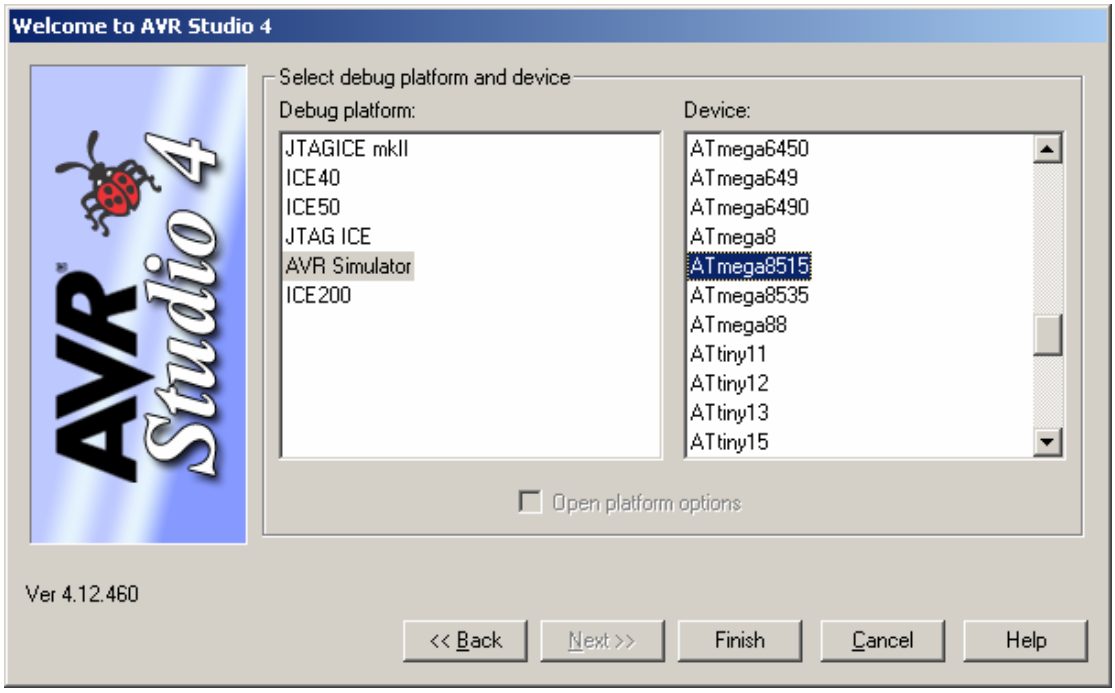


To create a new project, click on **New Project** (new Projects can also be created later by selecting **Project\New** from the Menu system). On the next dialog Box, select Atmel AVR assembler, enter the project name (eg "**first**") and navigate to your desired location by clicking the button labelled "**…**".

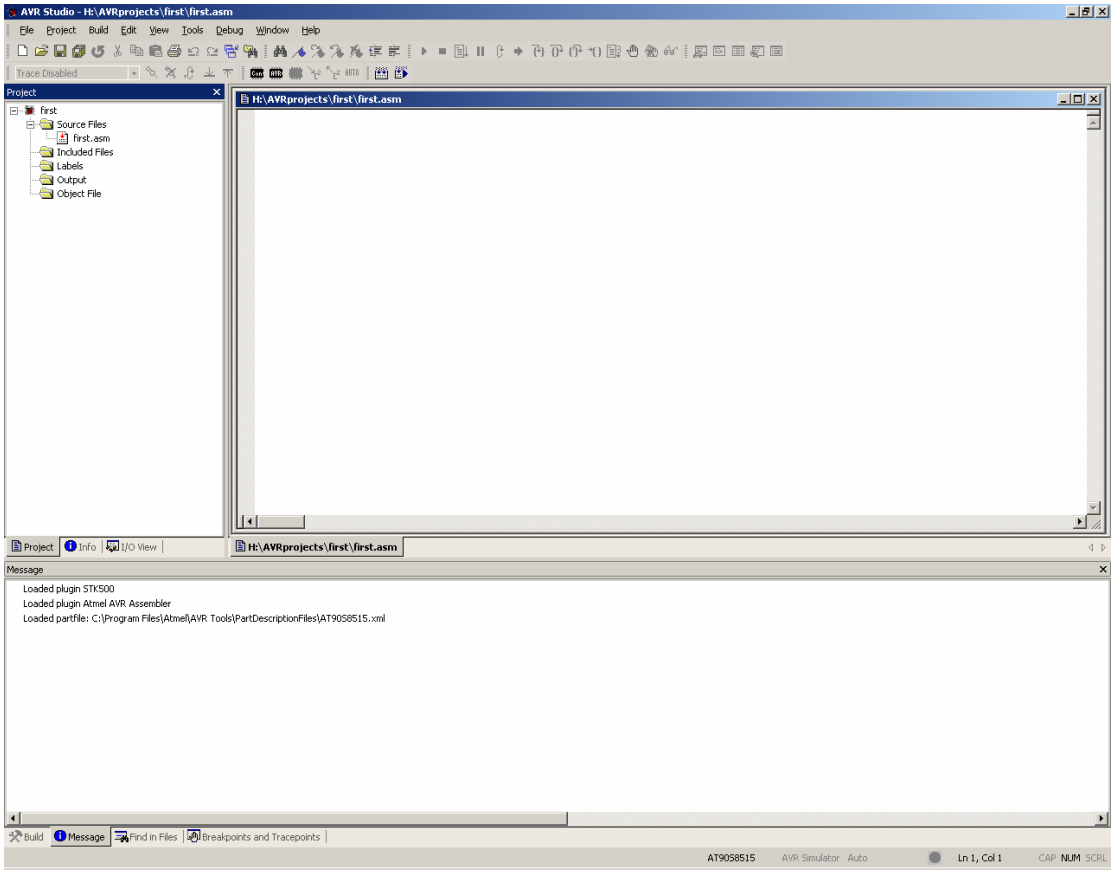
Your screen should look like:



Click **Next**. Now select the debug platform as **AVR Simulator** with the device **ATmega8515** so our program will be run on the simulator. Your screen should look like this:



Now click **Finish** and you will be shown a screen like this:



In your first.asm editor, copy and paste in the following code:

```
;
; My first assembler program
;
.include "8515def.inc"     ; include the 8515 definition file
.def temp = r16            ; define a temporary register

;
; In this example, we will output values to PORTB
;
RESET:
       ; Let's set the Data Direction Register for PORTB (DDRB)
       ; (0 = input, 1 = output)
       ; pin nums: 76543210
       ;           ||||||||
       ;           VVVVVVVV
       ldi temp, 0b11111111       ; this could also be 0xFF
       out DDRB, temp             ; output the value to DDRB
       ldi temp, 0x01             ; load 1 into temp

LOOP:
       ; Now, we continually loop writing to output B
       ; followed by rotating left once, and loop back
       out PORTB, temp            ; output temp to PORTB
       rol temp                   ; rotate temp left
       rjmp LOOP                  ; jump back to LOOP
```

Then save your file (**File\Save** or **Ctrl-S**) and build it (**Build\Build** or **F7**). Your code should appear like this:

H:\AVRprojects\first\first.asm

```
;
; My first assembler program
;
.include "8515def.inc"   ; include the 8515 definition file
.def temp = r16          ; define a temporary register

;
; In this example, we will output values to PORTB
;
RESET:
    ; Let's set the Data Direction Register for PORTB (DDRB)
    ; (0 = input, 1 = output)
    ; pin nums: 76543210
    ;           ||||||||
    ;           VVVVVVVV
    ldi temp, 0b11111111     ; this could also be 0xFF
    out DDRB, temp           ; output the value to DDRB
    ldi temp, 0x01           ; load 1 into temp

LOOP:
    ; Now, we continually loop writing to output B
    ; followed by rotating left once, and loop back
    out PORTB, temp          ; output temp to PORTB
    rol temp                 ; rotate temp left
    rjmp LOOP                ; jump back to LOOP
```

And your build output should appear like this:

Build

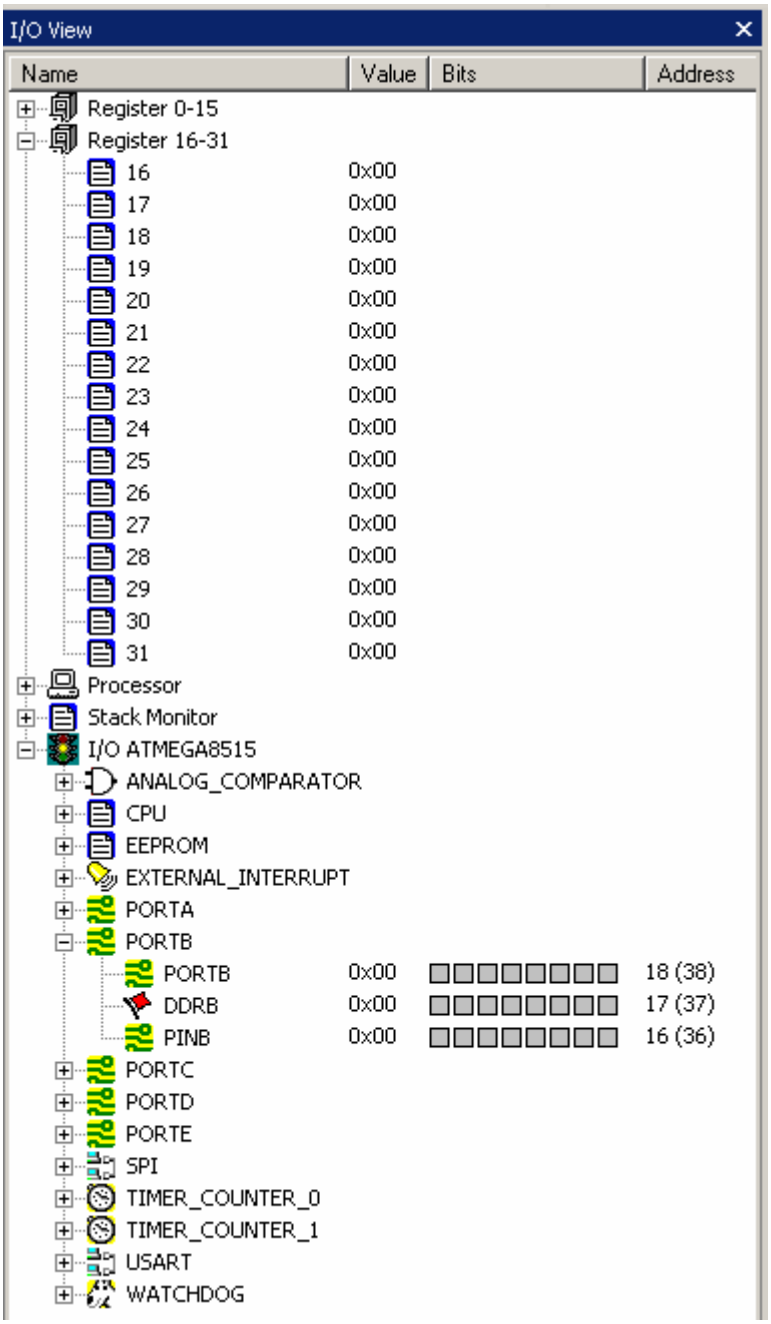
```
AVRASM: AVR macro assembler 2.1.2 (build 99 Nov  4 2005 09:35:05)
Copyright (C) 1995-2005 ATMEL Corporation

H:\AVRprojects\first\first.asm(4): Including file 'C:\Program Files\.
H:\AVRprojects\first\first.asm(24): No EEPROM data, deleting H:\AVRp:

AT90S8515 memory use summary [bytes]:
Segment   Begin    End      Code   Data   Used    Size   Use%
---------------------------------------------------------------
[.cseg] 0x000000 0x00000c     12      0     12    8192   0.1%
[.dseg] 0x000060 0x000060      0      0      0     512   0.0%
[.eseg] 0x000000 0x000000      0      0      0     512   0.0%

Assembly complete, 0 errors. 0 warnings
```

Notice the code has been built with no errors or warnings and that the code is 12 bytes in size (six assembler instructions at two bytes each).

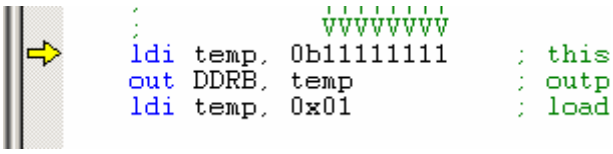
Using the AVR Simulator to test your first program

Now we are ready to simulate the code. In order to see what the code is doing, we need a better 'view'. So click the **I/O View** tab at the bottom of the docked window pane. Expand the **Register 16-31** tree and the **I/O ATMEGA8515** tree and then expand **PORTB**.

Your I/O View panel should look like this:



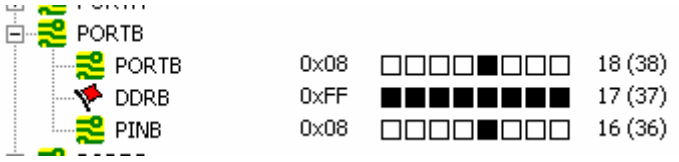
Now we are ready to start simulating. Select **Debug\Start Debugging** (or Ctrl-Alt-Shift-F5). You will now see a yellow arrow pointing to the next instruction to be executed, similar to this picture:



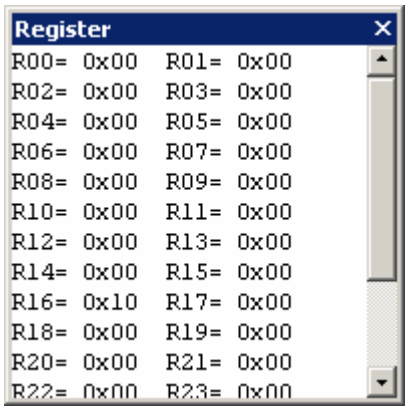
To step through each instruction, select **Debug\Step Into** (or press **F11**). Do this once now. The yellow arrow should be pointing to the next instruction (the "out" instruction). Before moving to the next instruction, check also the register list. You should see that the value for r16 has changed from 0x00 to 0xFF.

Press **F11** again. Now the DDRB value in the *I/O View* tree has changed to 0xFF, which is also shown by the dark squares. Each dark square represents one pin, from pin7 to pin0 (left to right). This means you have successfully written 0xFF (or 0b11111111 to DDRB).

Pressing **F11** a couple more times sets first register 16 and then PORTB to be the value 0x01. Next to PORTB, you will see 7 white squares (pins 7 to 1) and one dark square (pin 0). As you successively press **F11**, you will see the dark pin on PORTB shift left. This is what your *I/O View* screen might look like after a few iterations:

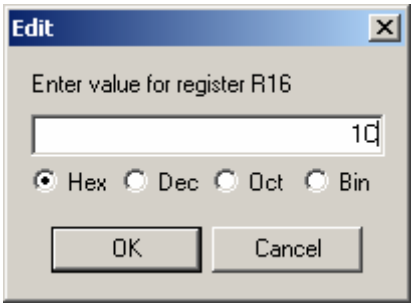


Now, let's say we wanted to further debug the program by editing the value in register 16? No matter what the current value of r16 is, let's give it the value 0x1C. To do this, open up the Register view (**View\Register** or **Alt-0**) and you should see a screen like this:

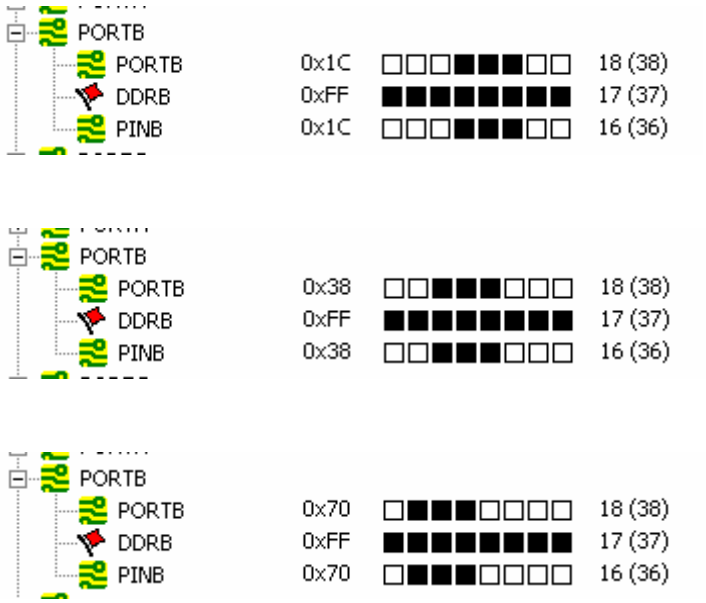


In order to change a value, double-click on the value for the register you need. In this case, **double-click on the value next to R16**. (In the picture above, you would click on the number 0x10).

You should see a screen like this:



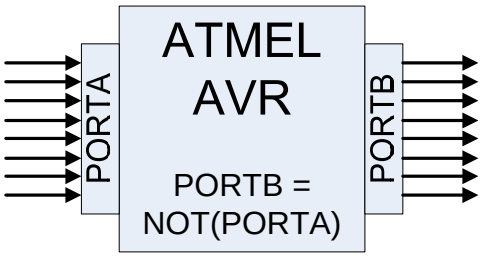
Enter in the value **1C** as above, and click **OK**. This change should be reflected in red in your Register View window as well as having changed the value of r16 under your I/O Tree View. Clicking F11 a few more times gives you these screen shots:



In order to stop debugging and return to editing your code, select **Debug\Stop Debugging** (or **Ctrl-Shift-F5**).

Simulating Inputs

What if, rather than continually outputting values, our AVR microcontroller was used to read values from one port and output to another. For example, we could use the AVR in this way:



Create a new project, or change the code in the current project. Copy and paste in the following code:

```
;
; My second assembler program
;
.include "8515def.inc"     ; include the 8515 definition file
.def temp = r16            ; define a temporary register

;
; Continually read in from PORTA and write out to PORTB
;
RESET:
       ; Let's set the Data Direction Registers (DDRA & DDRB)
       ; (0's = inputs, 1's = outputs)
       ldi    temp, 0x00
       out    DDRA, temp
       ldi    temp, 0xFF
       out    DDRB, temp

LOOP:
       ; Now, we continually loop, reading from PORTA pins,
       ; negating the value and writing to PORTB
       in     temp, PINA                ; read in from PORTA's input pins
       neg    temp                      ; negate temp register
       out    PORTB, temp               ; write out to PORTB
       rjmp   LOOP                      ; jump back to LOOP
```

Save (**Ctrl-S**) and Build (**F7**) your new project. Your assembly should complete with no errors again, but this time, the code size should display as 16 bytes (8 instructions at 2 bytes each).

```
AT90S8515 memory use summary [bytes]:
Segment   Begin    End      Code   Data   Used    Size   Use%
----------------------------------------------------------------
[.cseg] 0x000000 0x000010     16      0     16    8192   0.2%
[.dseg] 0x000060 0x000060      0      0      0     512   0.0%
[.eseg] 0x000000 0x000000      0      0      0     512   0.0%

Assembly complete, 0 errors. 0 warnings
```

Now, let's set up your I/O View to see both PortA and PortB. Collapse registers r16-r31, and expand Port A. Your view should look like this:

Select **Debug\Start Debugging** and press **F11** to step through the code and iterate the loop a couple of times. Regardless of the number of loop iterations you step through, your screen will stay in the following state:

That is, Port A's pins are constantly inputting 0x00, and thus Port B is always outputting 0xFF. In order to test your program, you need to simulate various inputs at the PINA pins. So, while your program is running, create a test bit-pattern on the PINA pins by clicking on each individual square. For example, the diagram below shows the bit pattern 0-1-0-1-1-0-1-0 (or 0x5A). Once you've entered this, complete one whole iteration of the loop by repeatedly pressing **F11**. Your screen should look like this:

Try testing other bit patterns to confirm that your code will work regardless of what bit pattern is applied to the inputs pins of Port A. You can also select **Debug\AutoStep** (**Alt-F5**) to have the code automatically stepped through.

You may wish to investigate the other hardware peripherals available for inspection under your **I/O View tree**. These include configuration registers for the ADC, external interrupts, timers and even internal CPU registers such as the *Status Register* (SREG), or internal processor registers such as the *Program Counter*.