Code Examples

Do not cut and paste the code examples in this word document into AVR studio Integrated Development Environment (IDE).

They will not work, due to word embedding hidden control characters into the text.

Instead, useable code examples from this document are in a separate file on the module's Moodle page.

From this file, the examples can be cut and pasted into AVR studio.

Installing Software on Home Machine

AVR Studio Download and install AVR Studio 4.19

Technical Support

CMS Technical Support can be contacted by email

cms-support@gre.ac.uk

AVR Studio

AVR Studio is an Integrated Development Environment (IDE) for writing and debugging AVR applications in Microsoft's windows environments.

The program, which can be downloaded free of charge from Atmel, includes a project management tool, source file editor, simulator, assembler and front-end for C/C++, programming, emulation, and on-chip debugging.

The full data sheets, ATmega8535 Datasheet and ATmega8535 Instruction Set for the device are on the module's Moodle page.

You will need to refer to these to complete this laboratory.

Example

Consider the following program.

```
     ldi   r16,$04  ;Line 1, Put 04 HEX into register r16
     ldi   r17,$06  ;Line 2, Put 06 HEX into register r17
     ldi   r18,$01  ;Line 3, Put 01 HEX into register r18
     ldi   r19,$F8  ;Line 4, Put F8 HEX into register r19
     add   r16,r17  ;Line 5, Add contents of r17 to r16, result to r16
     sub   r16,r18  ;Line 6, Subtract r18 from r16, result to r16
     add   r16,r19  ;Line 7, Add the contents of r16 to r19, result to r16

end: rjmp end       ;loop forever
```

TASK 1

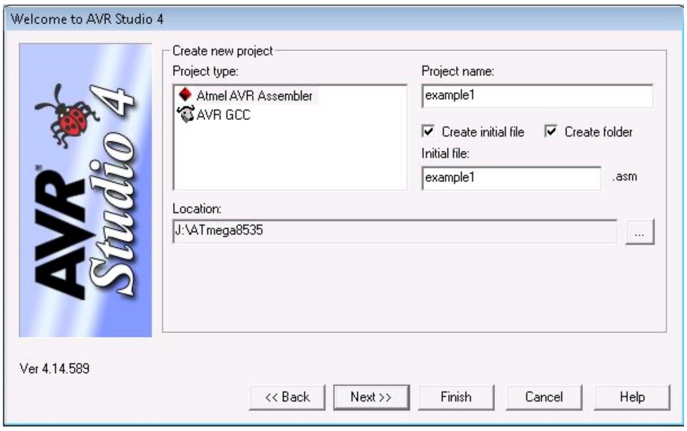
Examine the above program and complete the following table, showing the contents of each register after each line of the above program, is executed.

	r16	r17	r18	r19
Line 1	04	-	-	-
Line 2	04	06	-	-
Line 3	04	06	01	-
Line 4	04	06	01	F8
Line 5	0A	06	01	F8
Line 6	09	06	01	F8
Line 7	01	06	01	F8

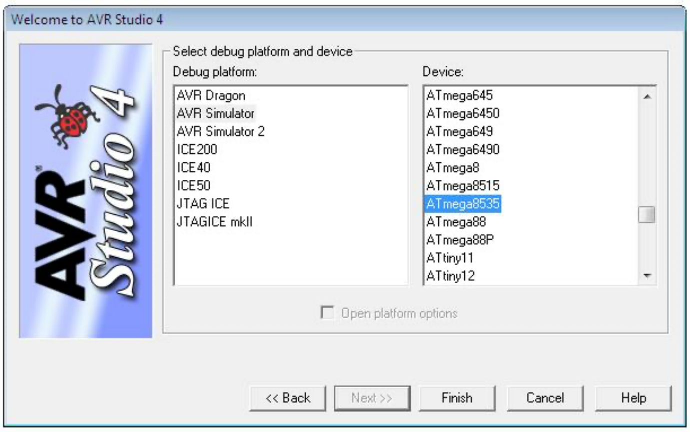
AVR Studio - Assembler

AVR Studio is project based, so you need to create a project before you can do anything.

1. Start AVR Studio
2. Run AVR Studio and select New Project
3. Choose Atmel AVR Assembler as Project Type
4. Ensure that 'Create initial file' and 'Create folder' are ticked
5. Give the Project a name
6. Set the Location of where to save the Project files



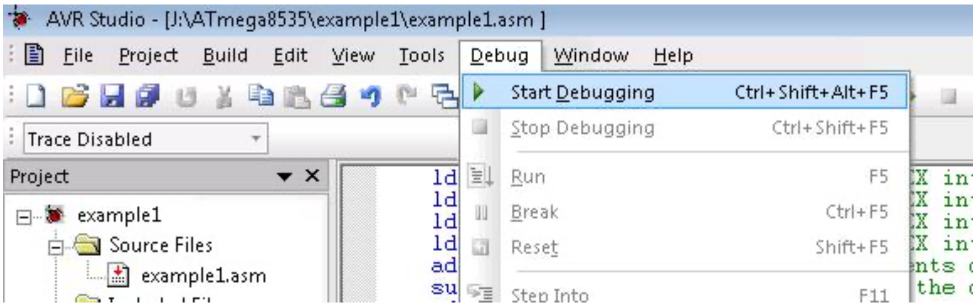
7. Click Next Button - On the next screen
8. Set Debug platform to AVR Simulator
9. Set Device to ATmega8535



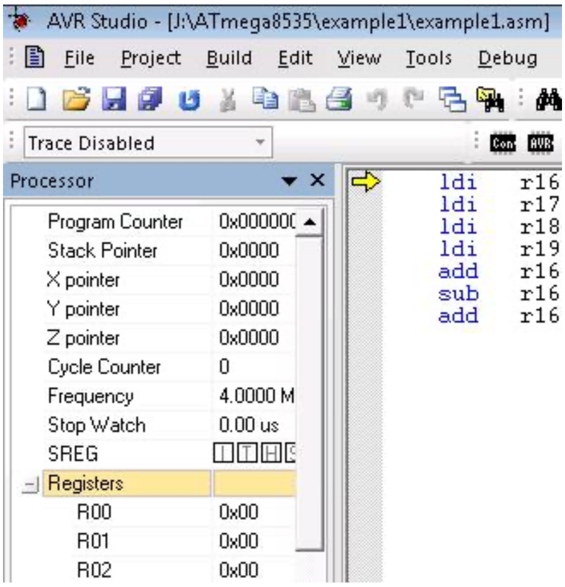
10. Click Finish Button
11. Cut and paste the program, from TASK 1 into the AVR window, and select 'Build'. The program should now assemble and indicate no errors. The assembled program can now be executed in AVR Studio's simulator.

AVR Studio Simulator

1. Select Debug > Start Debugging



2. Expose the registers in the processor window



TASK 2

Single step (F10) through the program and complete the table. The contents of the registers are not the same as in TASK 1. Why not?

	r16	r17	r18	r19
Line 1	04	-	-	-
Line 2	04	06	-	-
Line 3	04	06	01	-
Line 4	04	06	01	F8
Line 5	0A	06	01	F8
Line 6	09	06	01	F8
Line 7	01	06	01	F8

Assembling a Program with Errors

With any program there are two types of error.

- A syntax error which stops the program assembling.
- A logic error, the program produces the wrong answer.

TASK 3

The program below should calculate 3a + 2b - c where a=4, b=3, c=19.

Calculate what the answer should be. Answer___-1_______

```
;Program to calculate 3a + 2b - c
.equ    a     =3
.equ    b     =4
.equ    c     =19

      ldi   r16,a
      ldi   r17,b
      ldi   r18,c

;use register r20 to calculate 3a
      ldi   r20,$0
      add   r20,r16
      add   r20,r16
      add   r2O,r16
      add   r20,r16

;use register r21 to calculate 2b
      add   r21,r17
      ldd   r21,$0

;add 3a to 2b and put the result in r20
      sub   r21,r20

;put c into r22 then take it from the total in r20
      ldi   r22,c
      take  r21,r22

end:  rjmp  end       ;loop forever
```

Before running the debugger, the answer that we have calculated was -1 in decimal. However, after the running the debugger, the answer has shown as FF in Hexadecimal which is 255 in decimal.

The reason behind is that, the registers can store until 255. Once it hits 256 it gets rolled over and starts from 00 again.

Therefore, -1 stands one step behind the 0 which is the maximum value a register contain which is 255 (FF).

TASK 4

Assemble the program and correct the syntax errors.

Explain the relationship of the answer produced by the simulator to the answer you calculated.

Status Register

The Status Register contains information about the result of the most recently executed instruction. This information can be used for the flow of control within the program.

```
;flag test program

          ldi     r16,$80
          ldi     r17,$80
          add     r16,r17

          ldi     r16,$78
          ldi     r17,$63
          add     r16,r17

          ldi     r16,$fc
          ldi     r17,$f9
          add     r16,r17

          ldi     r16,252
          ldi     r17,249
          add     r16,r17

end:      rjmp    end       ;loop forever
```

TASK 5

Identify which flags are set in the status register and explain why the instruction / data caused each of the flags to be set.

To understand the process that is occurring, you need to notate the values of r16 and r17 in binary.

		r16	r17	I	T	H	S	V	N	Z	C	Explanation
ldi	r16,$80	10000000	-	-	-	-	-	-	-	-	-	No affect
ldi	r17,$80	10000000	10000000	-	-	-	-	-	-	-	-	No affect
add	r16,r17	00000000	10000000	0	0	0	1	1	0	1	1	
ldi	r16,$78	01111000	-									No affect
ldi	r17,$63	01111000	01100011									No affect
add	r16,r17	11011011	01100011	0	0	0	0	1	1	0	O	
ldi	r16,$FC	11111100	-									No affect
ldi	r17,$F9	11111100	11111011									No affect
add	r16,r17	11110101	11111011	0	0	1	1	0	1	0	1	
ldi	r16,252	11111100	-									No affect
ldi	r17,249	11111100	11111001									No affect
add	r16,r17	11110101	11111001	0	0	1	1	0	1	0	1	

Bitwise Operations

A bitwise operation takes two bit patterns of equal length and performs a logical operation, AND OR XOR, on the pair. NOT is a unary function.

- AND - For example 0101 AND 0011 = 0001
- OR - For example 0101 OR 0011 = 0111
- XOR - For example 0101 XOR 0011 = 0110
- NOT - For example NOT 0111 = 1000

TASK 6

Perform the following bitwise operations. To understand the process occurring, you need to notate the operand values in binary.

Operator	Operand 1	Operand 2	Answer
NOT	$A5	-	01011010
AND	$A5	$0F	00000101
OR	$A5	$0F	10101111
XOR	$A5	$0F	10101010

The ATmega8535 supports the bitwise operations of NOT, AND, OR and XOR with the following instructions: -

Operator	Instruction	Immediate Instruction
NOT	com	-
AND	and	andi
OR	or	ori
XOR	eor	-

Bit Manipulation

Bitwise operators are commonly used to manipulate bits within a register. Consider: -

To extract the four lower bits AND the register with a mask of 00001111, anything AND by 0 will be 0.	00101100	AND
	00001111	
	00001100	

To set the upper four bits to 1, OR the register with a mask of 11110000, anything OR by 1 will be 1.	00101100	OR
	11110000	
	11111100	

To flip the four lower bits, XOR the register with a mask of 00001111, anything XOR by 1, will be flipped.	00101100	XOR
	00001111	
	00100011	

TASK 7

Complete the following table

	Operand 1	Operator	Operand 2	Answer
Flip bits 4 and 5	10100101	XOR	00011000	10011101
Extract bits 1 and 5	10100101	AND	00010001	00000001
Set bits 4 and 6 to 1	10100101	OR	00101000	11001101
Set 1,2,3 and 7 bits to 1	10100101	OR	01000111	11100111

Shifting and Rolling

- `lsl   Rd   ;Logical Shift Left`
- `lsr   Rd   ;Logical Shift Right`
- `rol   Rd   ;Rotate Left Through Carry`
- `ror   Rd   ;Rotate Right Through Carry`
- `asr   Rd   ;Arithmetic Shift Right`

Assemble and single step the program, making note of the effect of each instruction on the number in r16.

To understand the process that is occurring, you need to notate the operand values in binary for each line of code.

```
;bit shifting and rolling

      ldi    r16,$0F
loop: lsl    r16     ;Logical Shift Left
      lsr    r16     ;Logical Shift Right
      rol    r16     ;Rotate Left Through Carry
      ror    r16     ;Rotate Right Through Carry
      asr    r16     ;Arithmetic Shift Right
      rjmp   loop
```

r16
00001111
00011110
00001111
00011110
00001111
00000111

TASK 8

Using the bitwise operations and the shift/rolling instructions.

Write a program to rearrange the bits as shown, the bar above means NOT. Refer to the Power Point Bitwise Example in the Moodle module page for help.

Start Position	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Final Position	$\overline{\text{bit 3}}$	$\overline{\text{bit 2}}$	bit 7	bit 6	bit 1	bit 0	$\overline{\text{bit 5}}$	$\overline{\text{bit 4}}$

Check your program by referring to the TASK 8 expected results, in the Moodle module page.

Branching

TASK 9

For each of the following program segments, examine the registers during execution to familiarise yourself with the operation of each of the branch instructions. Some loops will not exit – identify why.

```
      ldi  r16,9
loop: dec r16
      dec r16
      brne loop
end:  rjmp end

      ldi r16,9
loop: dec r16
      dec r16
      brmi loop
end:  rjmp end

      ldi r16,9
loop: dec r16
      dec r16
      brpl loop
end:  rjmp end

      ldi r16,0
```

```
loop: inc r16
      cpi r16,4
      breq loop
end:  rjmp end

      ldi r16,1
loop: inc r16
      cpi r16,5
      brne loop

      ldi r16,3
loop: inc r16
      cpi r16,6
      brne loop
end:  rjmp end

      ldi r16,1
loop: inc r16
      cpi r16,3
      breq next
next: rjmp loop
end:  rjmp end
```

Looping

Assemble and single step the following program.

```
;program to calculate 5+4+3+2+1 using a loop

      ldi   r16,0
      ldi   r17,5
loop: add   r16,r17
      dec   r17
      brne  loop

end:  rjmp   end     ;loop forever
```

TASK 10

Make the following changes to the above program

- Add together 7+6+5+4+3+2+1
- Add together 7+5+3+1

Assemble and single step the following program

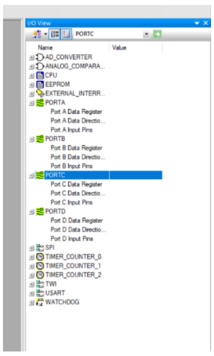
```
;Nested loop example
        ldi     r16,$00      ;Initialise counter
        ldi     r24,$03      ;Initialise 2nd loop counter
loop2:  ldi     r25,$02      ;Initialise 1st loop counter
loop1:  inc     r16          ;Increment counter
        dec     r25          ;Decrement the 1st loop counter
        brne    loop1        ;and continue to decrement until 1st loop
counter = 0
        dec     r24          ;Decrement the 2nd loop counter
        brne    loop2        ;If the 2nd loop counter is not equal to
zero repeat the 1st loop, else continue

end:    rjmp    end          ;loop forever
```

TASK 11

Modify the program, such that on completion r16 = $C8

Port Access

The AVR Studio simulator can be used to input values to the ports, and display port outputs. The ports are shown on the right side of the simulator.



Assemble and single step the following program, changing the input value on port A.

```
;Program to echo port A input to ports B and C

;Port Addresses
.equ    DDRA    =$1A       ;Port A Data Direction Register Address
```

```
.equ    PINA    =$19        ;Port A Input Address
.equ    DDRB    =$17        ;Port B Data Direction Register Address
.equ    PORTB   =$18        ;Port B Output Address
.equ    DDRC    =$14        ;Port C Data Direction Register Address
.equ    PORTC   =$15        ;Port C Output Address

;Register Definitions
.def    temp    =r16        ;Temporary storage register

;Program Initialisation
;Initialise Ports
        ldi     temp,$00
        out     DDRA,temp  ;Set Port A for input
        ldi     temp,$FF
        out     DDRB,temp  ;Set Port B for output
        out     DDRC,temp  ;Set Port C for output

;Main Program
loop:   in      r17,PINA   ;Read the value on port A.
        out     PORTC,r17  ;Output the value to port C.
        out     PORTB,r17  ;Output the value to port B.
        rjmp    loop       ;Repeat forever
```

TASK 12

Modify the program, to read in from port B and echo port B to port A and port C. You will need to look at the lecture.