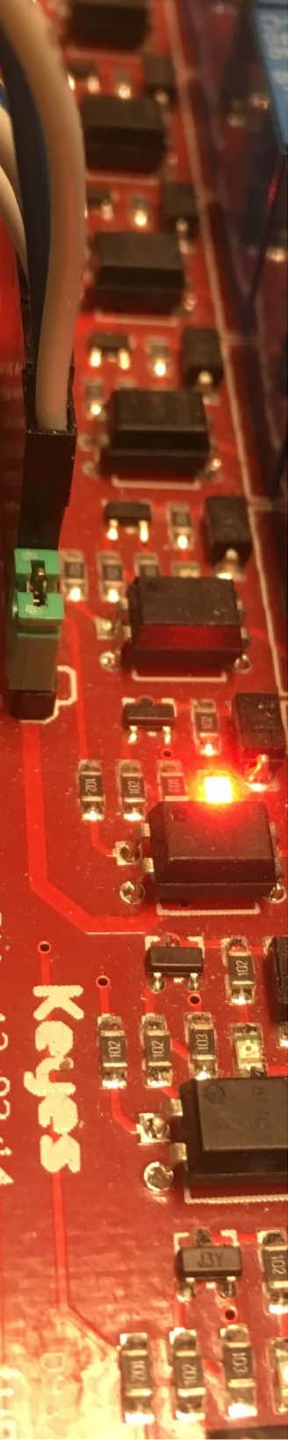
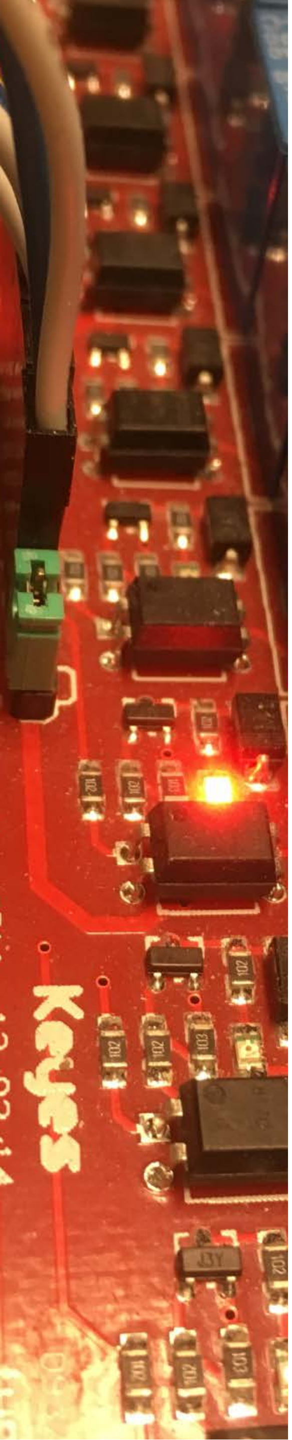


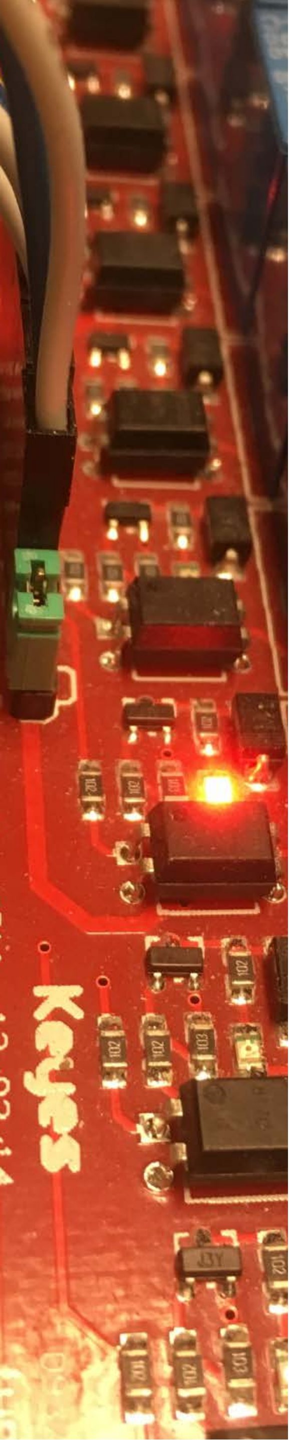
1DT301, Computer Technology
Lecture #3,

Program example,
some instructions,
assembler commands,
m2560def.inc.

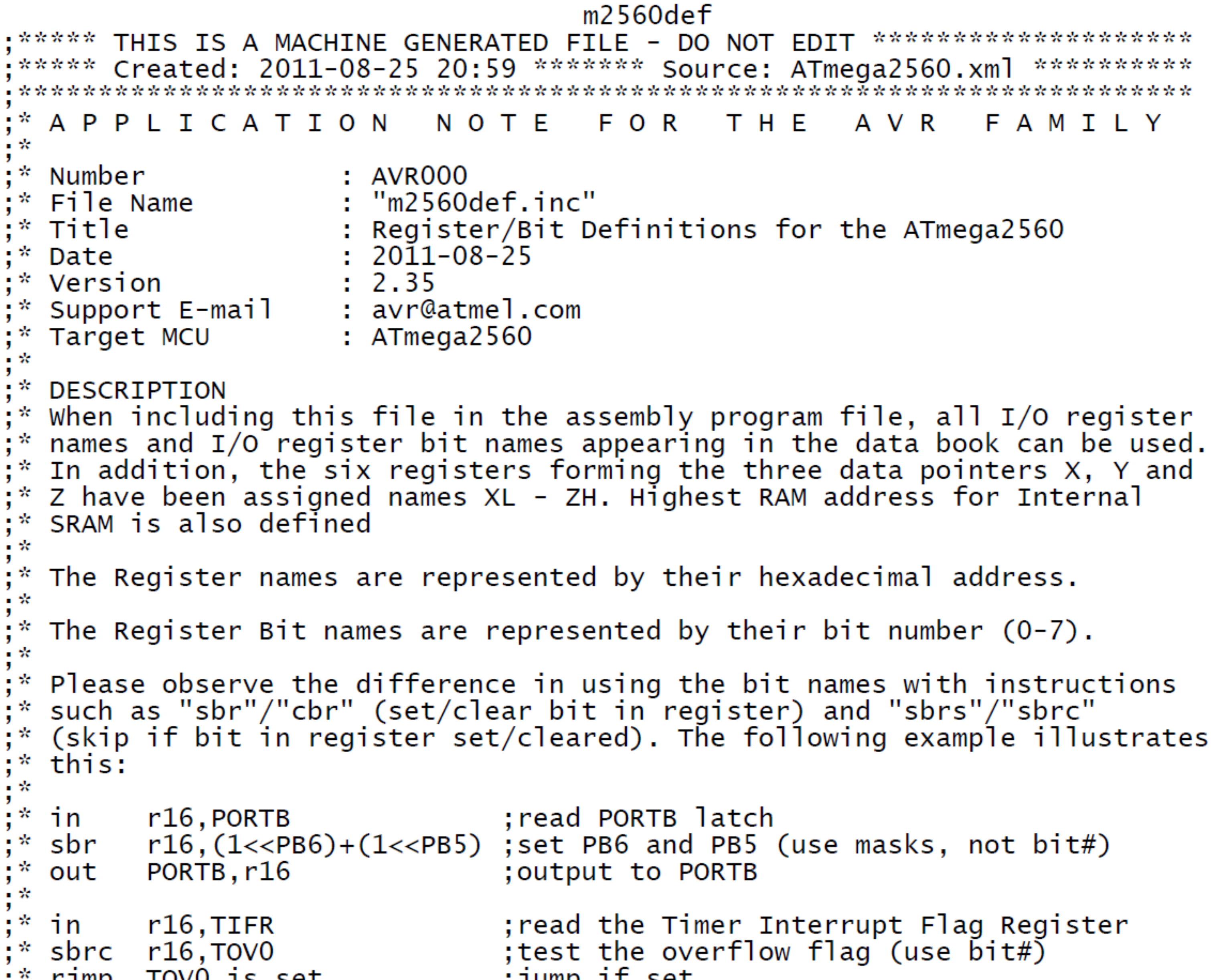


Before a lab

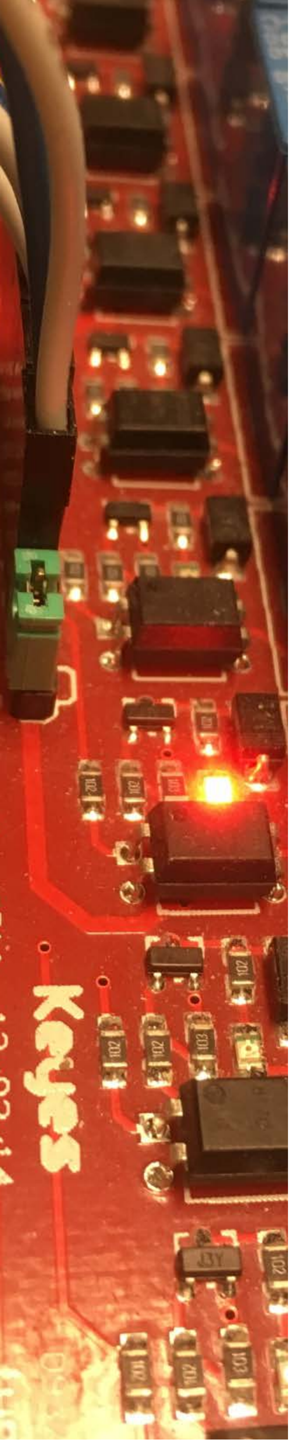
- Try to prepare as much as possible by
  - Creating a flow chart (a mental model)
  - Creating a project and add code
  - Running the simulator



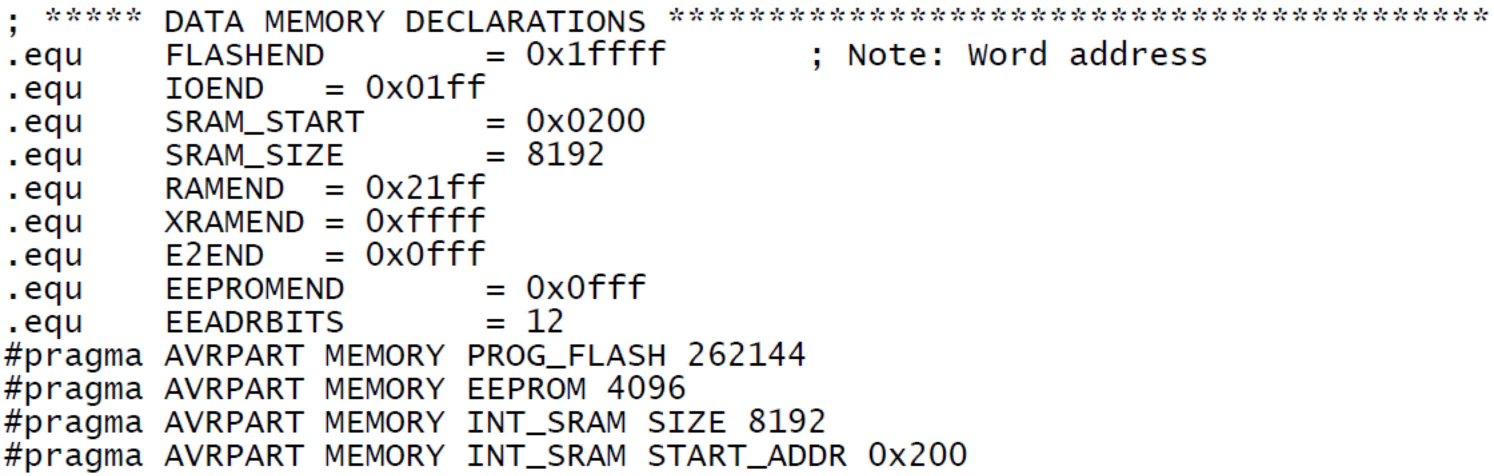
m2560def.inc



```
                                   m2560def
;***** THIS IS A MACHINE GENERATED FILE - DO NOT EDIT ********************
;***** Created: 2011-08-25 20:59 ******* Source: ATmega2560.xml **********
;*************************************************************************
;* A P P L I C A T I O N   N O T E   F O R   T H E   A V R   F A M I L Y
;*
;* Number              : AVR000
;* File Name           : "m2560def.inc"
;* Title               : Register/Bit Definitions for the ATmega2560
;* Date                : 2011-08-25
;* Version             : 2.35
;* Support E-mail      : avr@atmel.com
;* Target MCU          : ATmega2560
;*
;* DESCRIPTION
;* When including this file in the assembly program file, all I/O register
;* names and I/O register bit names appearing in the data book can be used.
;* In addition, the six registers forming the three data pointers X, Y and
;* Z have been assigned names XL - ZH. Highest RAM address for Internal
;* SRAM is also defined
;*
;* The Register names are represented by their hexadecimal address.
;*
;* The Register Bit names are represented by their bit number (0-7).
;*
;* Please observe the difference in using the bit names with instructions
;* such as "sbr"/"cbr" (set/clear bit in register) and "sbrs"/"sbrc"
;* (skip if bit in register set/cleared). The following example illustrates
;* this:
;*
;* in    r16,PORTB              ;read PORTB latch
;* sbr   r16,(1<<PB6)+(1<<PB5)  ;set PB6 and PB5 (use masks, not bit#)
;* out   PORTB,r16              ;output to PORTB
;*
;* in    r16,TIFR               ;read the Timer Interrupt Flag Register
;* sbrc  r16,TOV0               ;test the overflow flag (use bit#)
;* rjmp  TOV0_is_set            ;jump if set
```

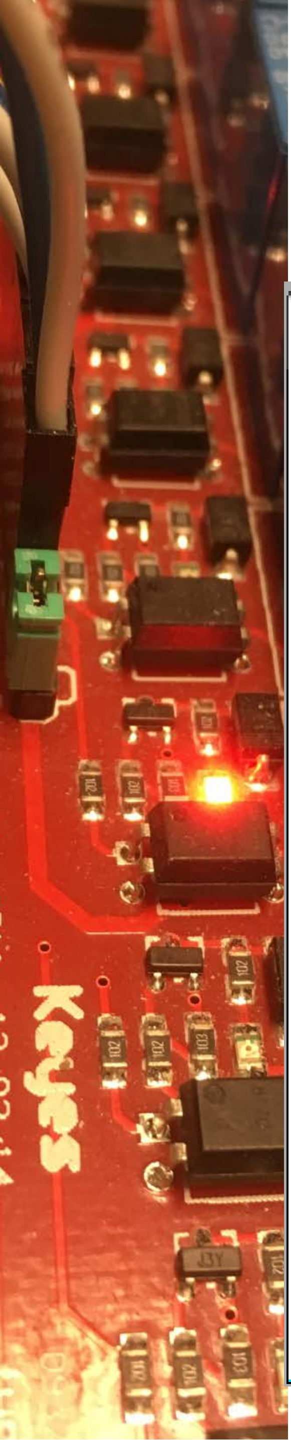


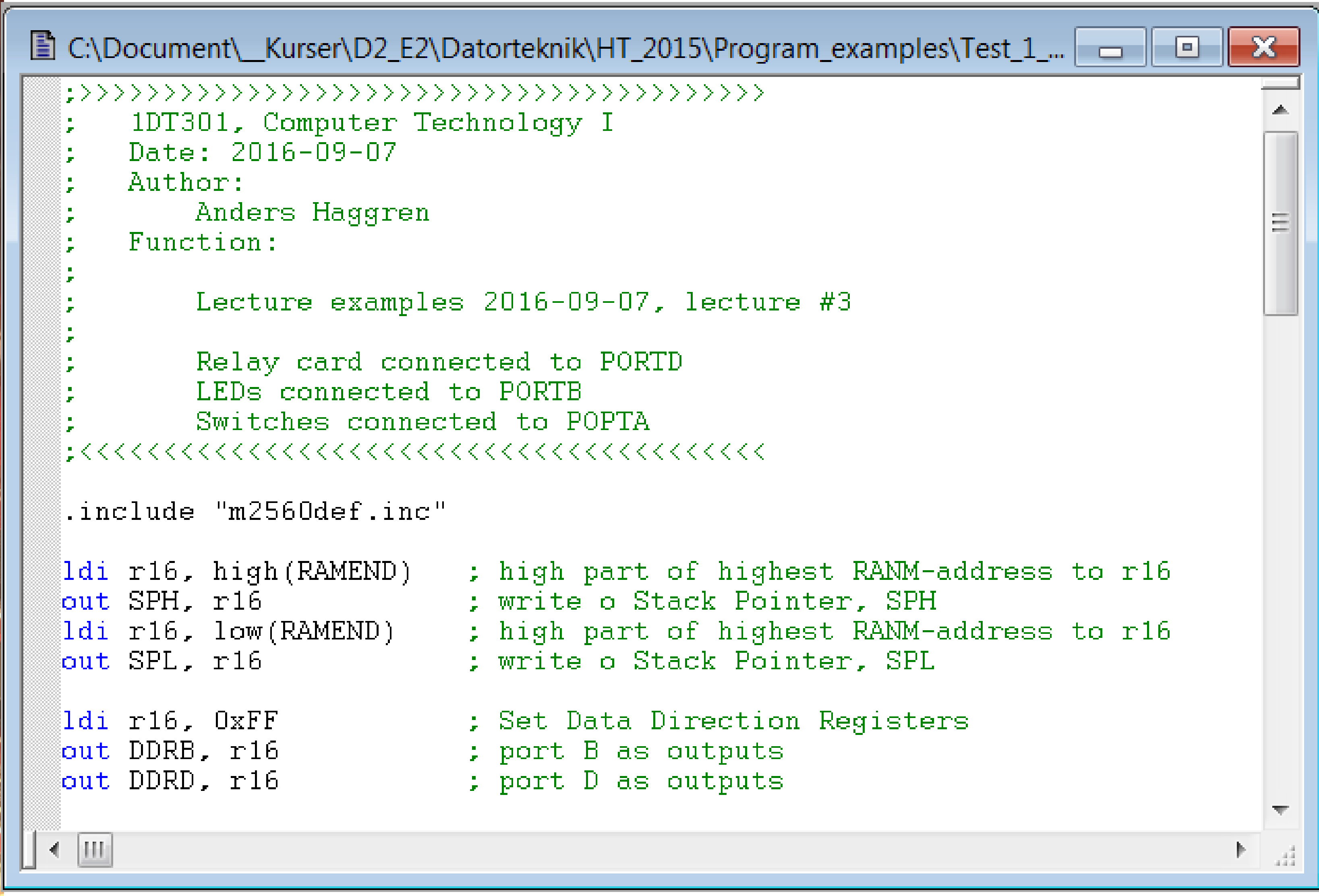
m2560def.inc



```
; ***** DATA MEMORY DECLARATIONS *****************************************
.equ    FLASHEND        = 0x1ffff       ; Note: Word address
.equ    IOEND   = 0x01ff
.equ    SRAM_START      = 0x0200
.equ    SRAM_SIZE       = 8192
.equ    RAMEND  = 0x21ff
.equ    XRAMEND = 0xffff
.equ    E2END   = 0x0fff
.equ    EEPROMEND       = 0x0fff
.equ    EEADRBITS       = 12
#pragma AVRPART MEMORY PROG_FLASH 262144
#pragma AVRPART MEMORY EEPROM 4096
#pragma AVRPART MEMORY INT_SRAM SIZE 8192
#pragma AVRPART MEMORY INT_SRAM START_ADDR 0x200
```

Program example

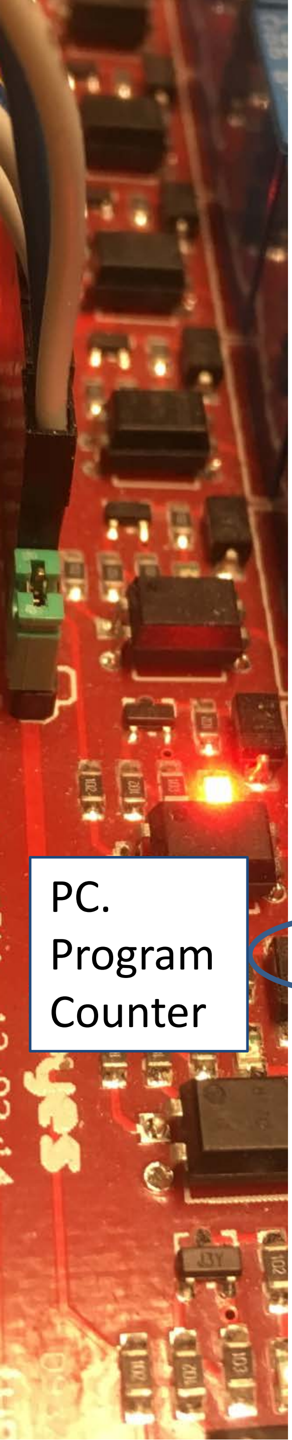


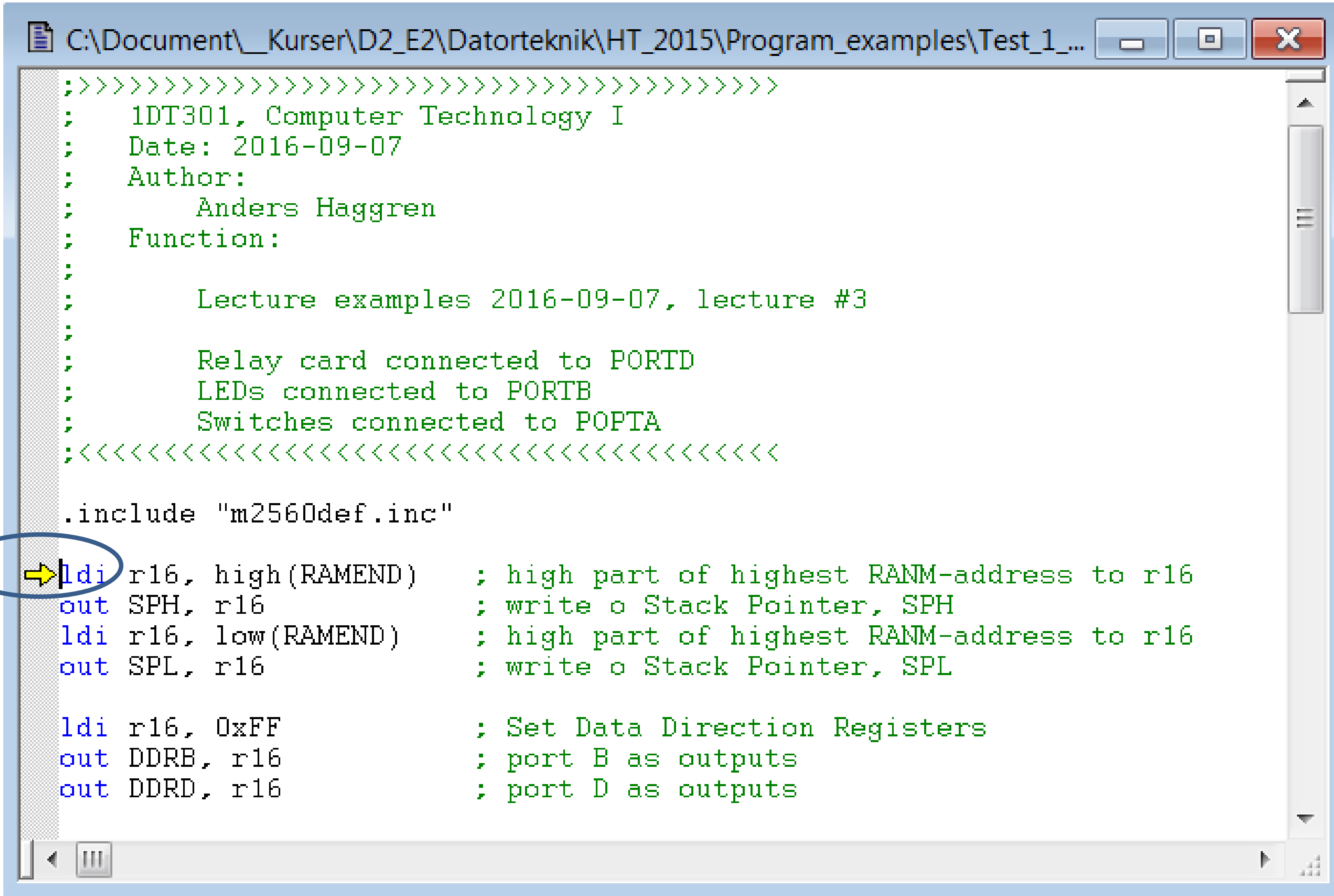


C:\Document\__Kurser\D2_E2\Datorteknik\HT_2015\Program_examples\Test_1_...

```
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;   1DT301, Computer Technology I
;   Date: 2016-09-07
;   Author:
;       Anders Haggren
;   Function:
;
;       Lecture examples 2016-09-07, lecture #3
;
;       Relay card connected to PORTD
;       LEDs connected to PORTB
;       Switches connected to POPTA
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

.include "m2560def.inc"

ldi r16, high(RAMEND)   ; high part of highest RANM-address to r16
out SPH, r16            ; write o Stack Pointer, SPH
ldi r16, low(RAMEND)    ; high part of highest RANM-address to r16
out SPL, r16            ; write o Stack Pointer, SPL

ldi r16, 0xFF           ; Set Data Direction Registers
out DDRB, r16           ; port B as outputs
out DDRD, r16           ; port D as outputs
```

Build and run the simulator

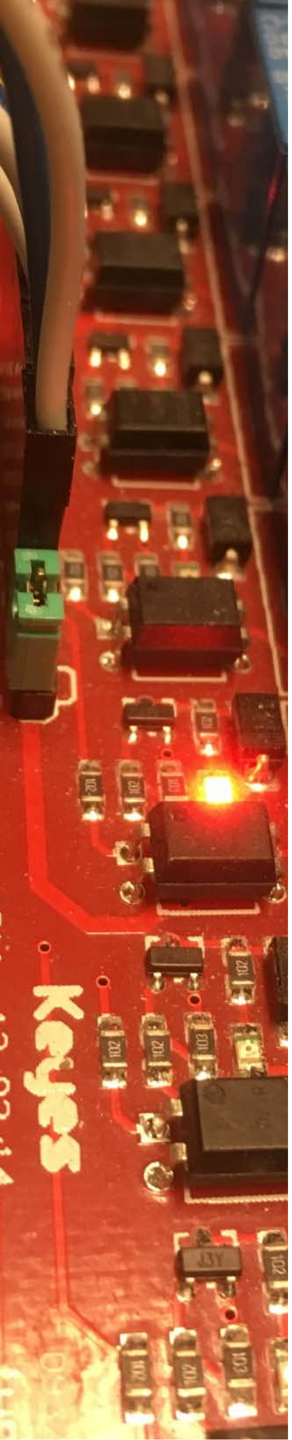




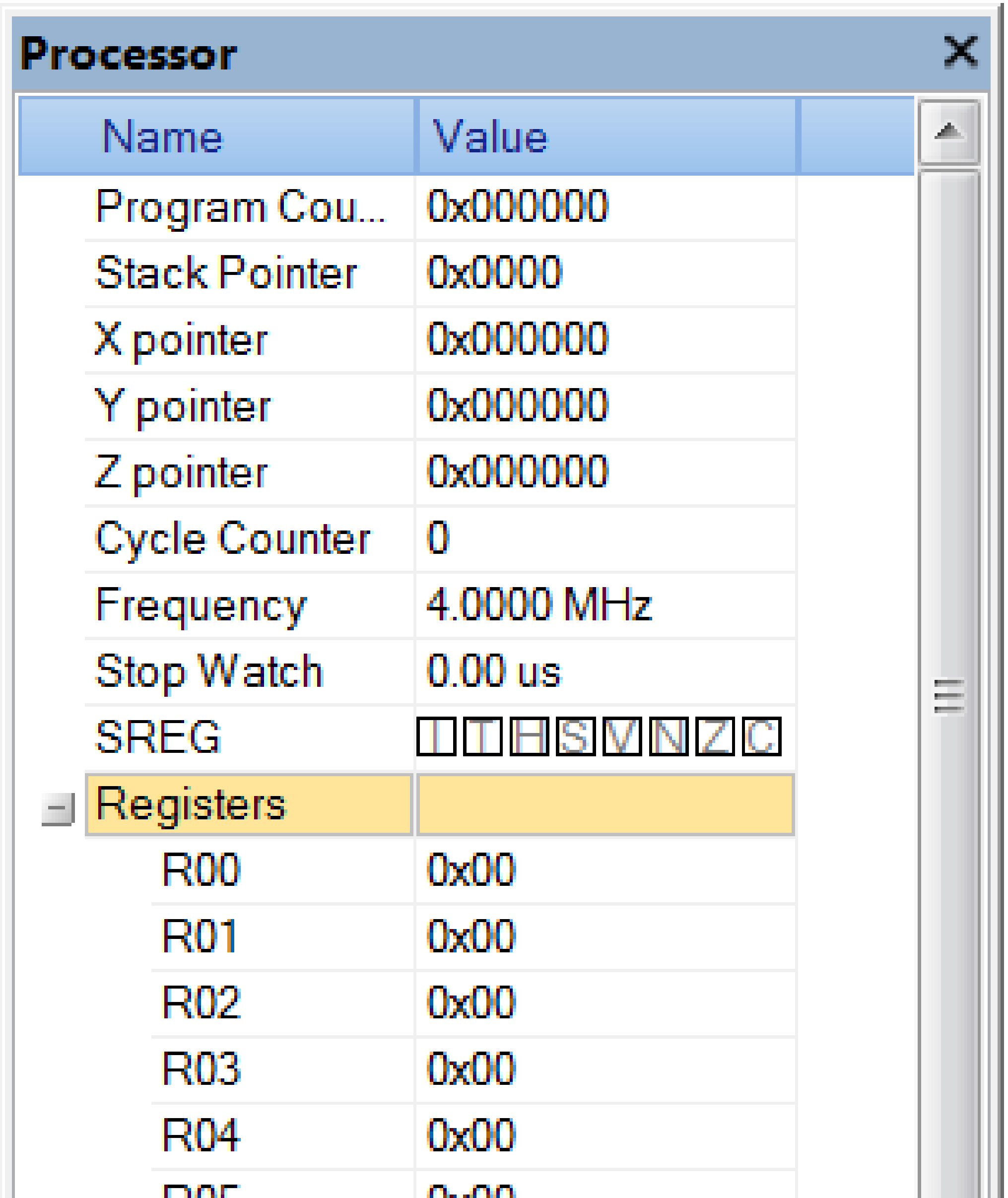
C:\Document_Kurser\D2_E2\Datorteknik\HT_2015\Program_examples\Test_1_...

```
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;    1DT301, Computer Technology I
;    Date: 2016-09-07
;    Author:
;        Anders Haggren
;    Function:
;
;        Lecture examples 2016-09-07, lecture #3
;
;        Relay card connected to PORTD
;        LEDs connected to PORTB
;        Switches connected to POPTA
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

.include "m2560def.inc"

ldi r16, high(RAMEND)      ; high part of highest RANM-address to r16
out SPH, r16               ; write o Stack Pointer, SPH
ldi r16, low(RAMEND)       ; high part of highest RANM-address to r16
out SPL, r16               ; write o Stack Pointer, SPL

ldi r16, 0xFF              ; Set Data Direction Registers
out DDRB, r16              ; port B as outputs
out DDRD, r16              ; port D as outputs
```

PC.
Program
Counter



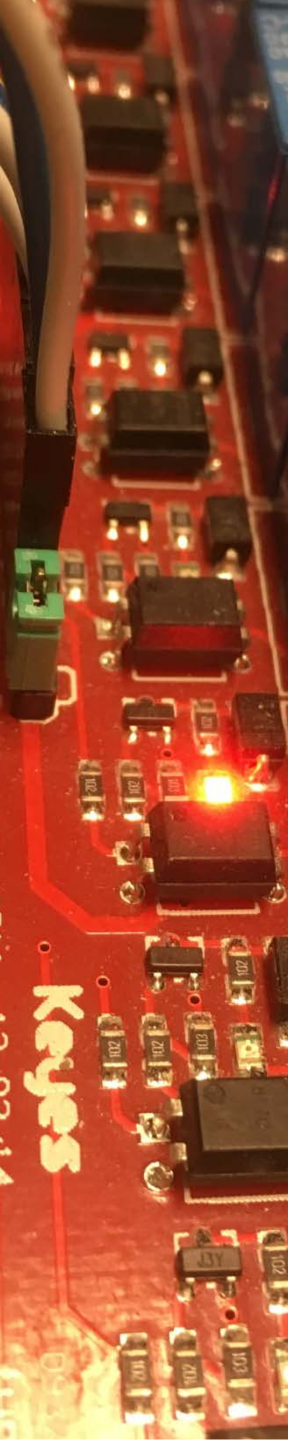
Processor Window, PC and SP

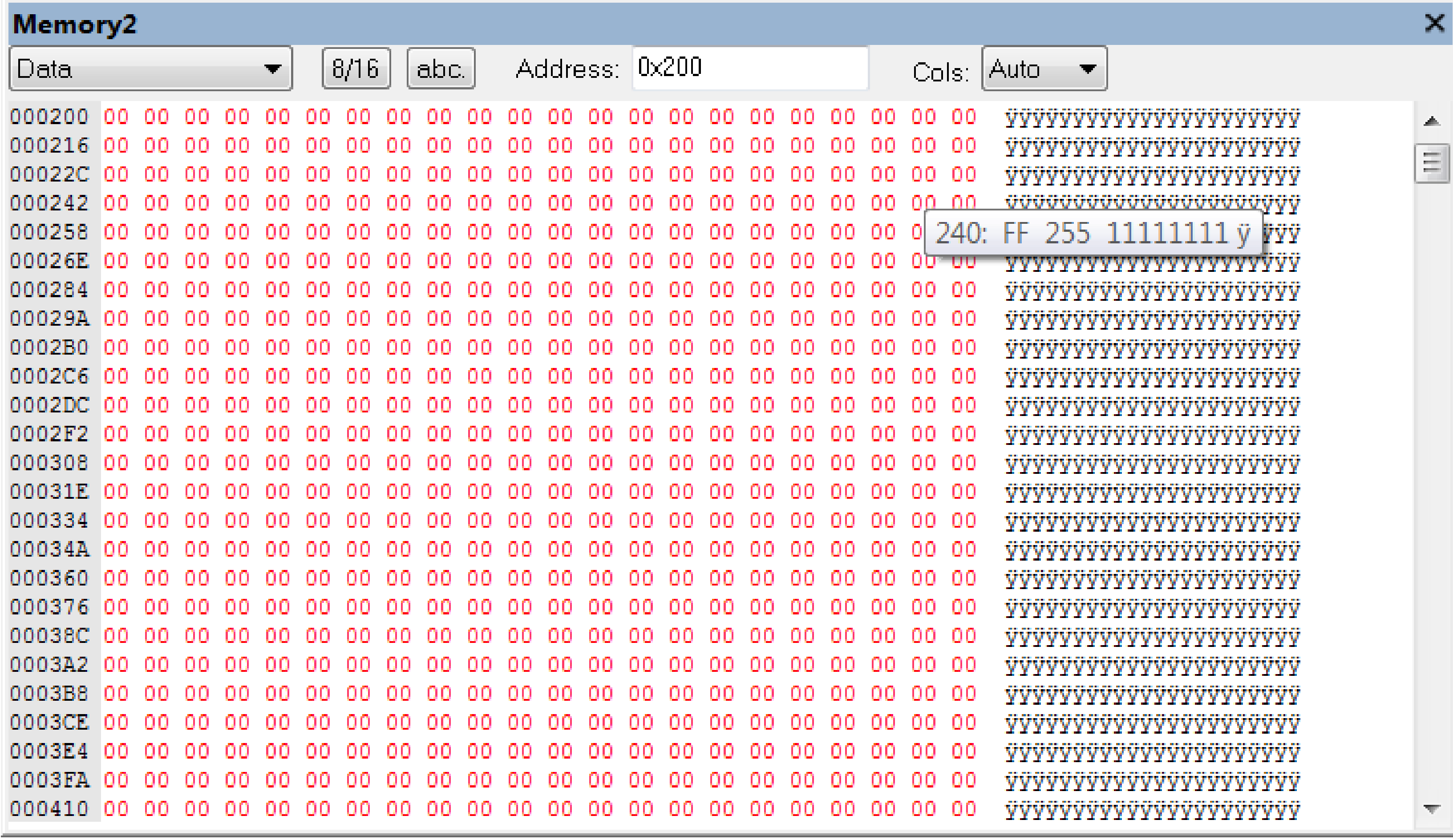


Processor

Name	Value
Program Cou...	0x000000
Stack Pointer	0x0000
X pointer	0x000000
Y pointer	0x000000
Z pointer	0x000000
Cycle Counter	0
Frequency	4.0000 MHz
Stop Watch	0.00 us
SREG	I T H S V N Z C
Registers	
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00

Data Memory



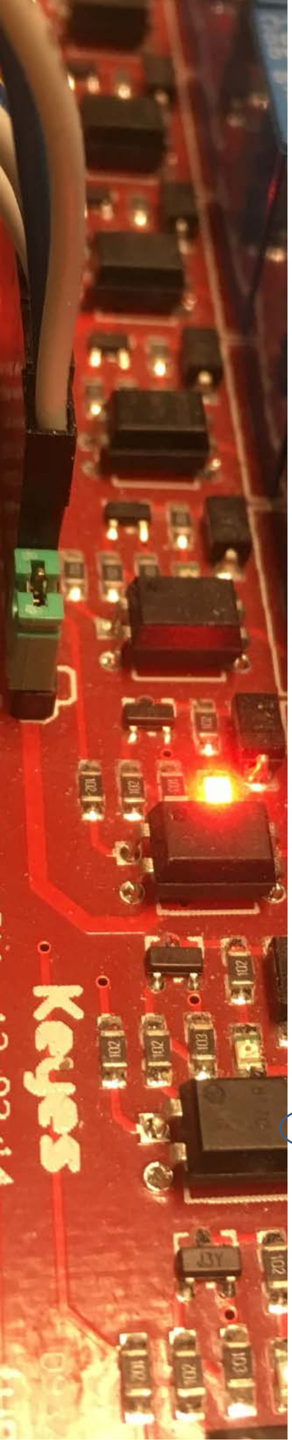


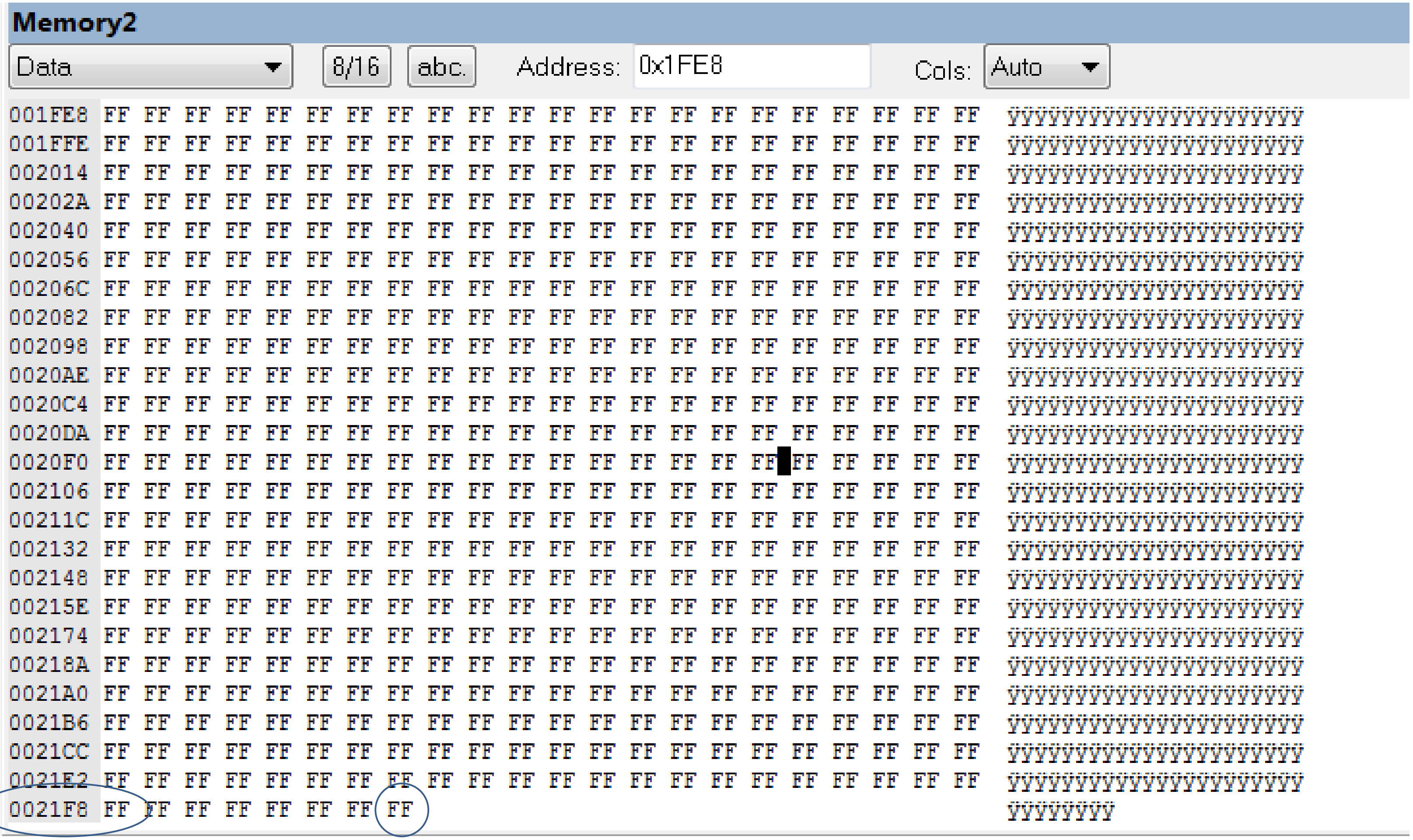
Memory2

Data ▼ | 8/16 | abc. | Address: 0x200 | Cols: Auto ▼

```
000200 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000216 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00022C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000242 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000258 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0 240: FF 255 11111111 ÿ ÿÿÿ
00026E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000284 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00029A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0002B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0002C6 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0002DC 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0002F2 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000308 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00031E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000334 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00034A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000360 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000376 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00038C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0003A2 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0003B8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0003CE 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0003E4 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0003FA 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000410 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
```

Data Memory



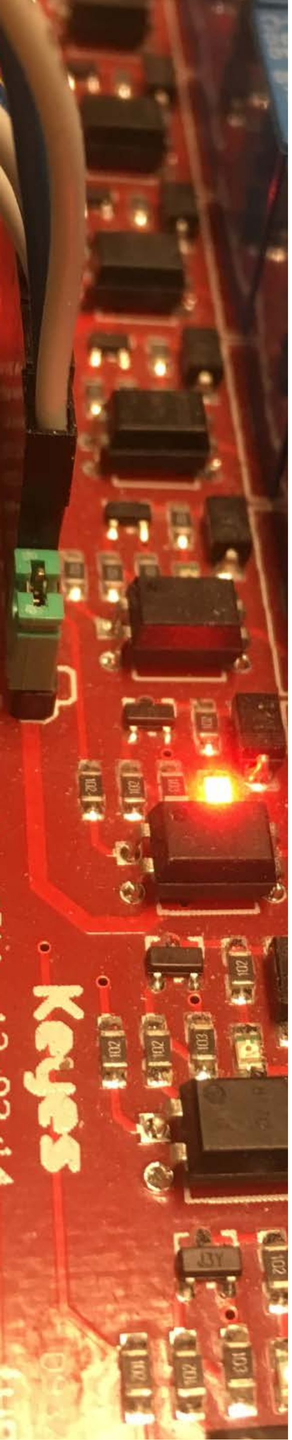


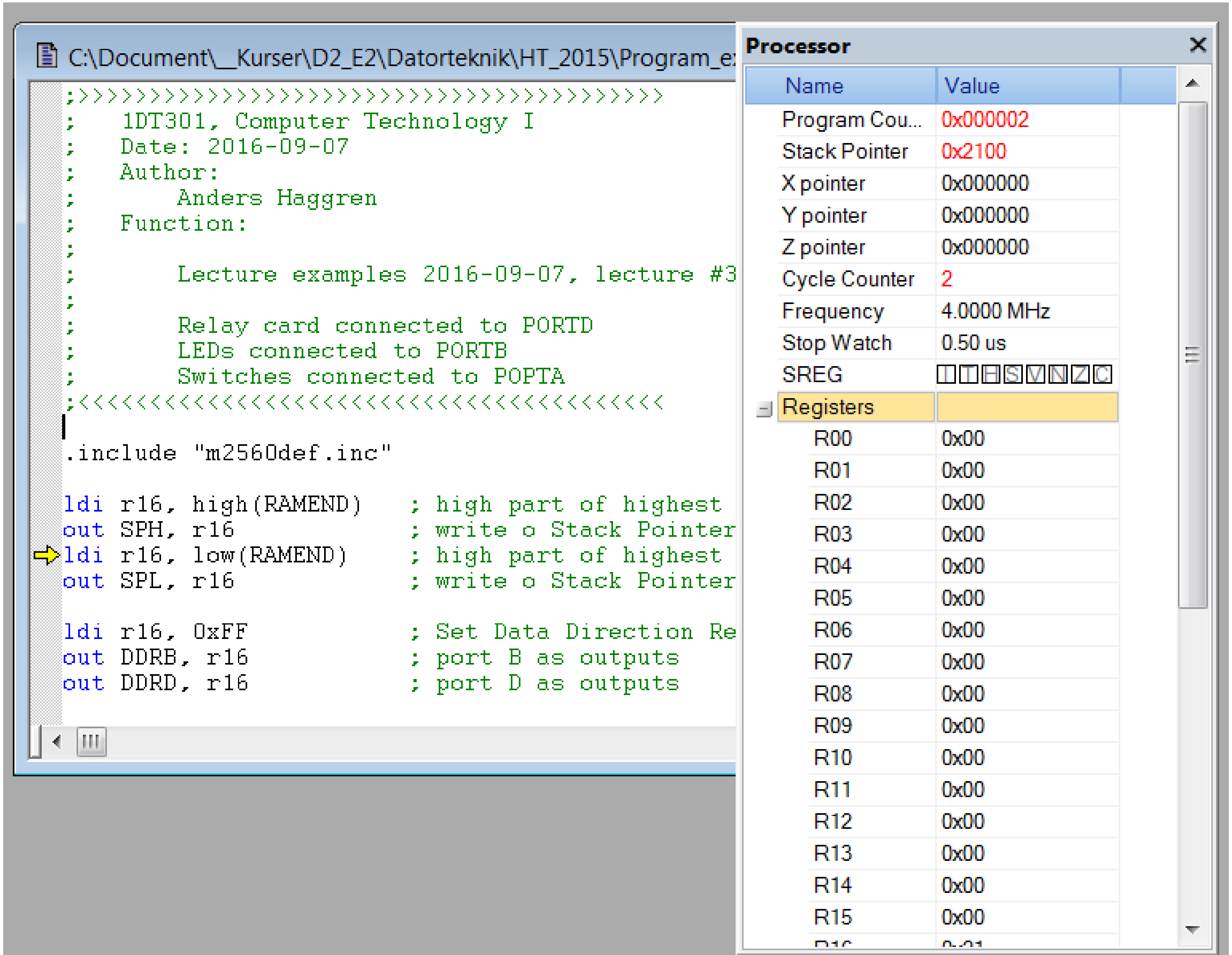
Memory2

Data | 8/16 | abc. | Address: 0x1FE8 | Cols: Auto

```
001FE8 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
001FFE FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002014 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00202A FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002040 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002056 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00206C FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002082 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002098 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020AE FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020C4 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020DA FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020F0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002106 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00211C FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002132 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002148 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00215E FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002174 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00218A FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021A0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021B6 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021CC FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021E2 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021F8 FF FF FF FF FF FF FF FF                                            ÿÿÿÿÿÿÿÿ
```

21F8, 21F9, 21FA, 21FB, 21FC, 21FD, 21FE, 21FF



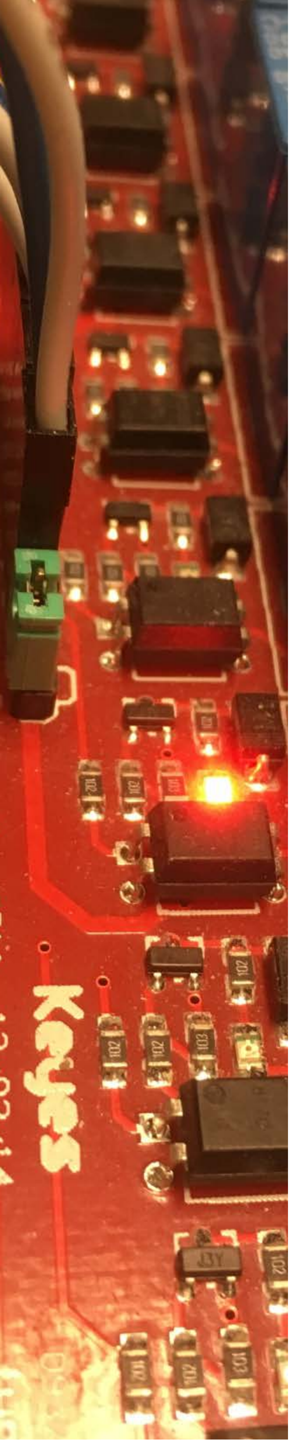


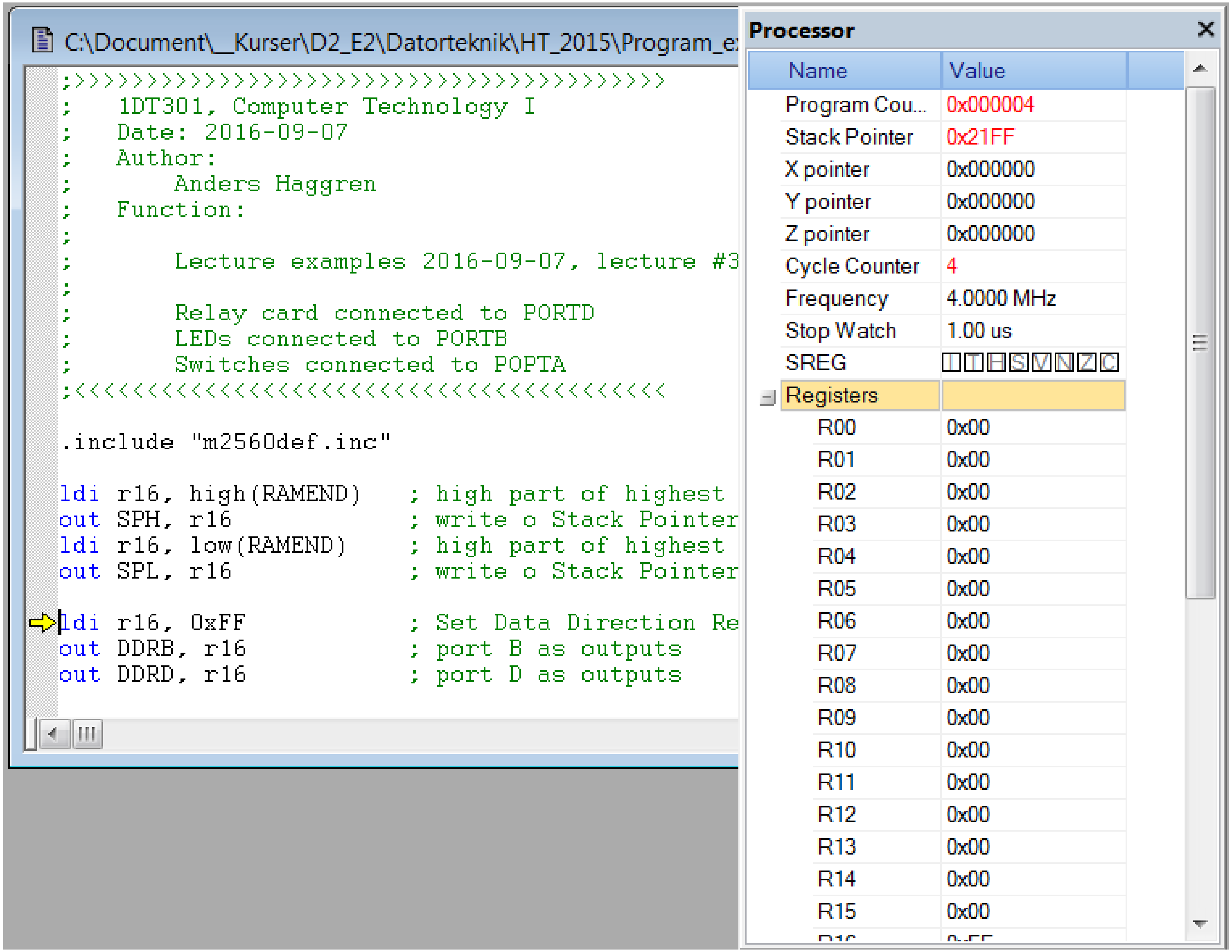
C:\Document_Kurser\D2_E2\Datorteknik\HT_2015\Program_e

```
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;    1DT301, Computer Technology I
;    Date: 2016-09-07
;    Author:
;        Anders Haggren
;    Function:
;
;        Lecture examples 2016-09-07, lecture #3
;
;        Relay card connected to PORTD
;        LEDs connected to PORTB
;        Switches connected to POPTA
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

.include "m2560def.inc"

ldi r16, high(RAMEND)    ; high part of highest
out SPH, r16             ; write o Stack Pointer
ldi r16, low(RAMEND)     ; high part of highest
out SPL, r16             ; write o Stack Pointer

ldi r16, 0xFF            ; Set Data Direction Re
out DDRB, r16            ; port B as outputs
out DDRD, r16            ; port D as outputs
```

Processor

Name	Value
Program Cou...	0x000002
Stack Pointer	0x2100
X pointer	0x000000
Y pointer	0x000000
Z pointer	0x000000
Cycle Counter	2
Frequency	4.0000 MHz
Stop Watch	0.50 us
SREG	I T H S V N Z C
Registers	
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x01



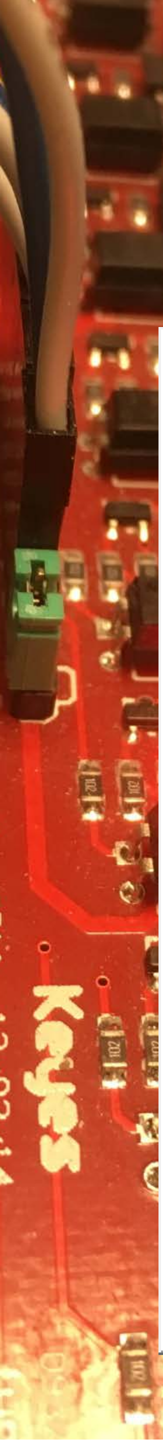


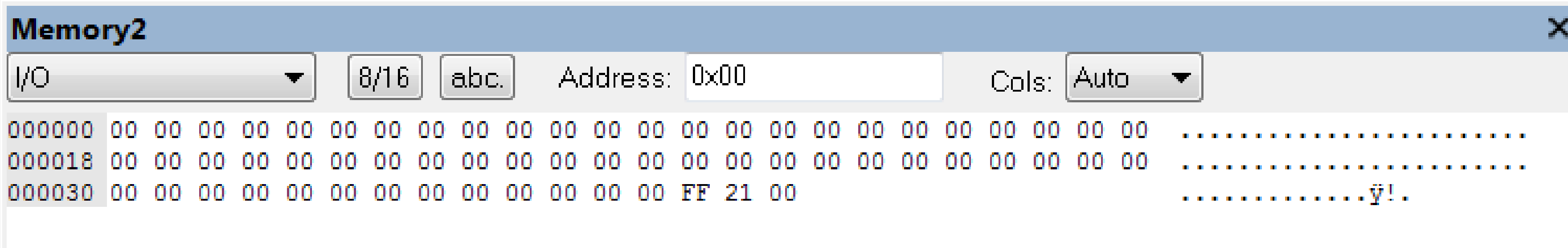
C:\Document\__Kurser\D2_E2\Datorteknik\HT_2015\Program_e

```
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;   1DT301, Computer Technology I
;   Date: 2016-09-07
;   Author:
;       Anders Haggren
;   Function:
;
;       Lecture examples 2016-09-07, lecture #3
;
;       Relay card connected to PORTD
;       LEDs connected to PORTB
;       Switches connected to POPTA
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

.include "m2560def.inc"

ldi r16, high(RAMEND)   ; high part of highest
out SPH, r16            ; write o Stack Pointer
ldi r16, low(RAMEND)    ; high part of highest
out SPL, r16            ; write o Stack Pointer

ldi r16, 0xFF           ; Set Data Direction Re
out DDRB, r16           ; port B as outputs
out DDRD, r16           ; port D as outputs
```

Processor

Name	Value
Program Cou...	0x000004
Stack Pointer	0x21FF
X pointer	0x000000
Y pointer	0x000000
Z pointer	0x000000
Cycle Counter	4
Frequency	4.0000 MHz
Stop Watch	1.00 us
SREG	I T H S V N Z C
Registers	
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0xFF



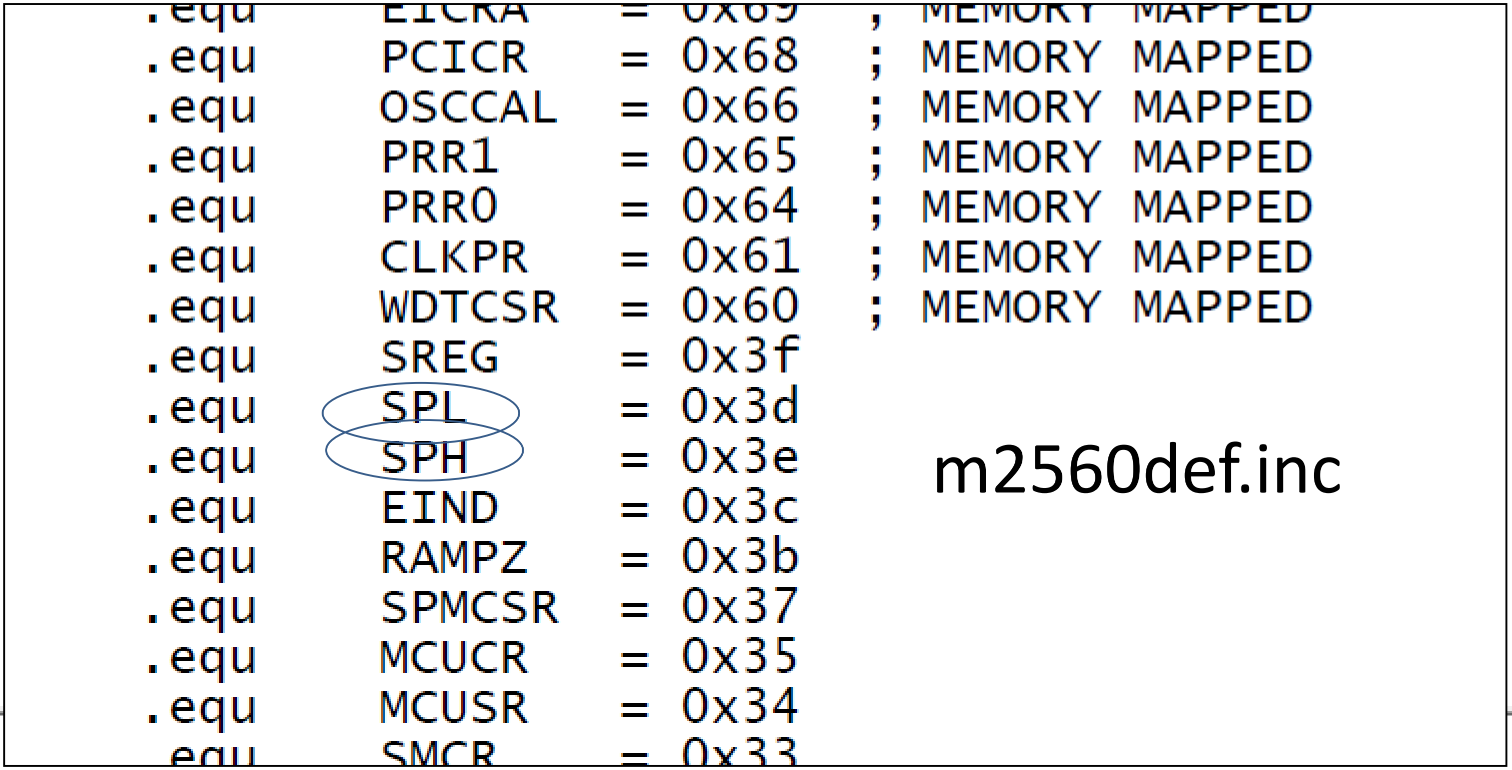


I/O

Address: 0x00

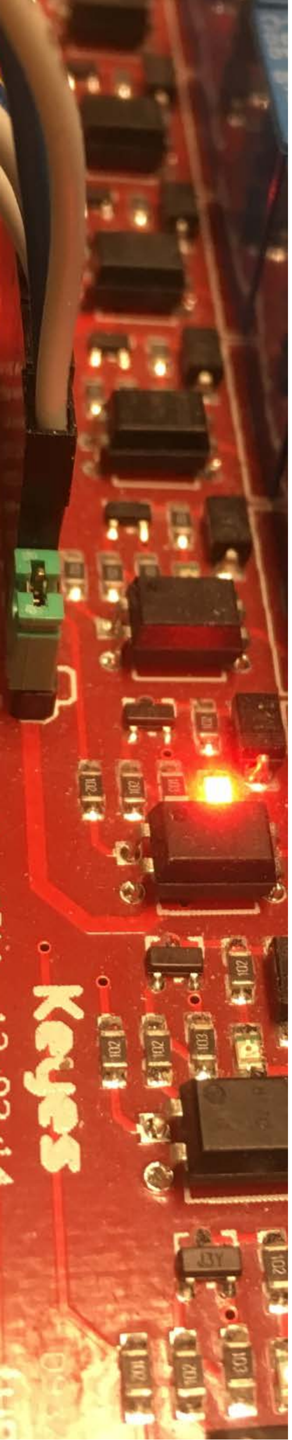
Cols: Auto

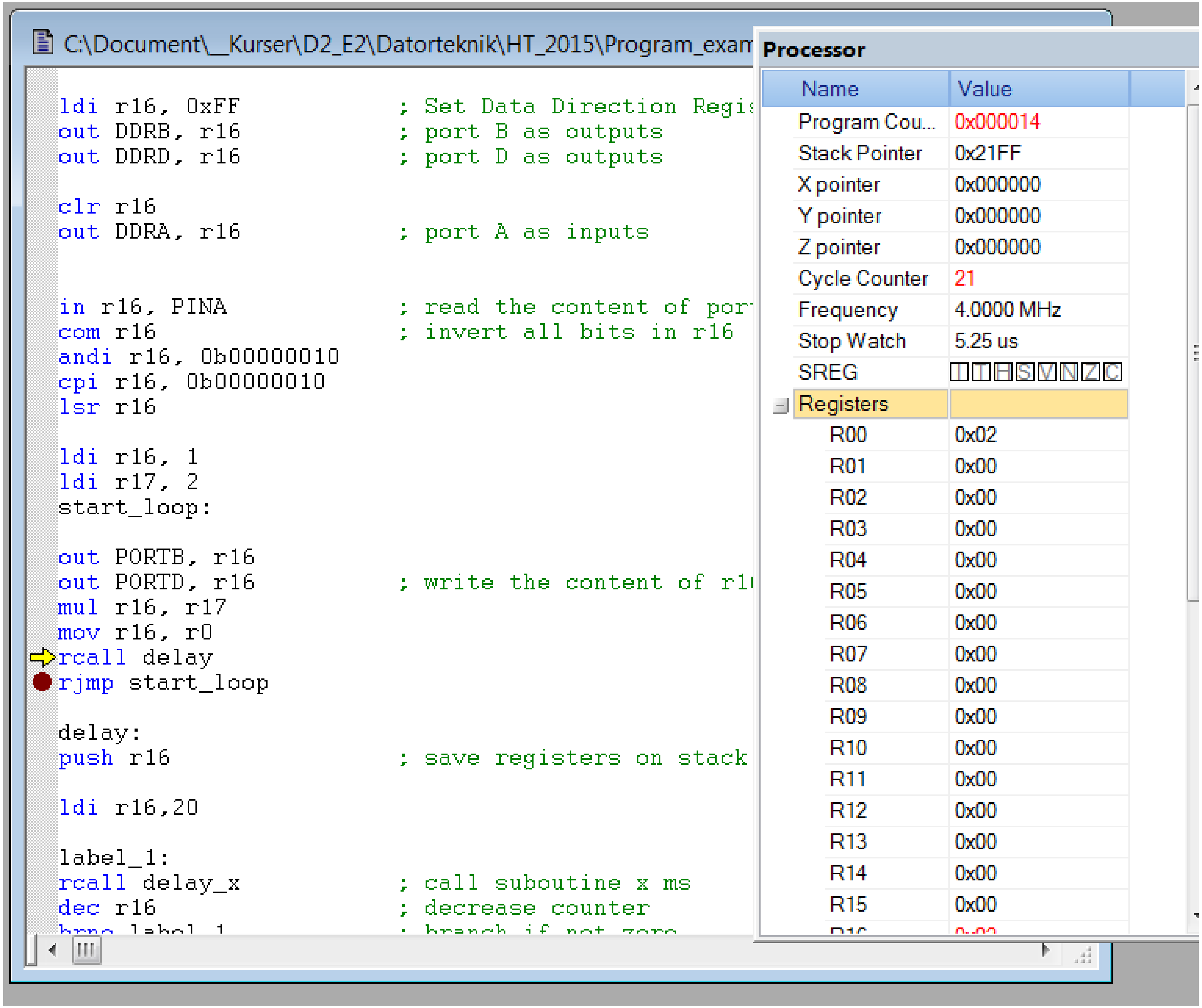
```
000000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ........................
000018 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ........................
000030 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 21 00                          .............ÿ!.
```



```
    .equ    EICRA   = 0x69  ; MEMORY MAPPED
    .equ    PCICR   = 0x68  ; MEMORY MAPPED
    .equ    OSCCAL  = 0x66  ; MEMORY MAPPED
    .equ    PRR1    = 0x65  ; MEMORY MAPPED
    .equ    PRR0    = 0x64  ; MEMORY MAPPED
    .equ    CLKPR   = 0x61  ; MEMORY MAPPED
    .equ    WDTCSR  = 0x60  ; MEMORY MAPPED
    .equ    SREG    = 0x3f
    .equ    SPL     = 0x3d
    .equ    SPH     = 0x3e
    .equ    EIND    = 0x3c
    .equ    RAMPZ   = 0x3b
    .equ    SPMCSR  = 0x37
    .equ    MCUCR   = 0x35
    .equ    MCUSR   = 0x34
    .equ    SMCR    = 0x33
```

m2560def.inc



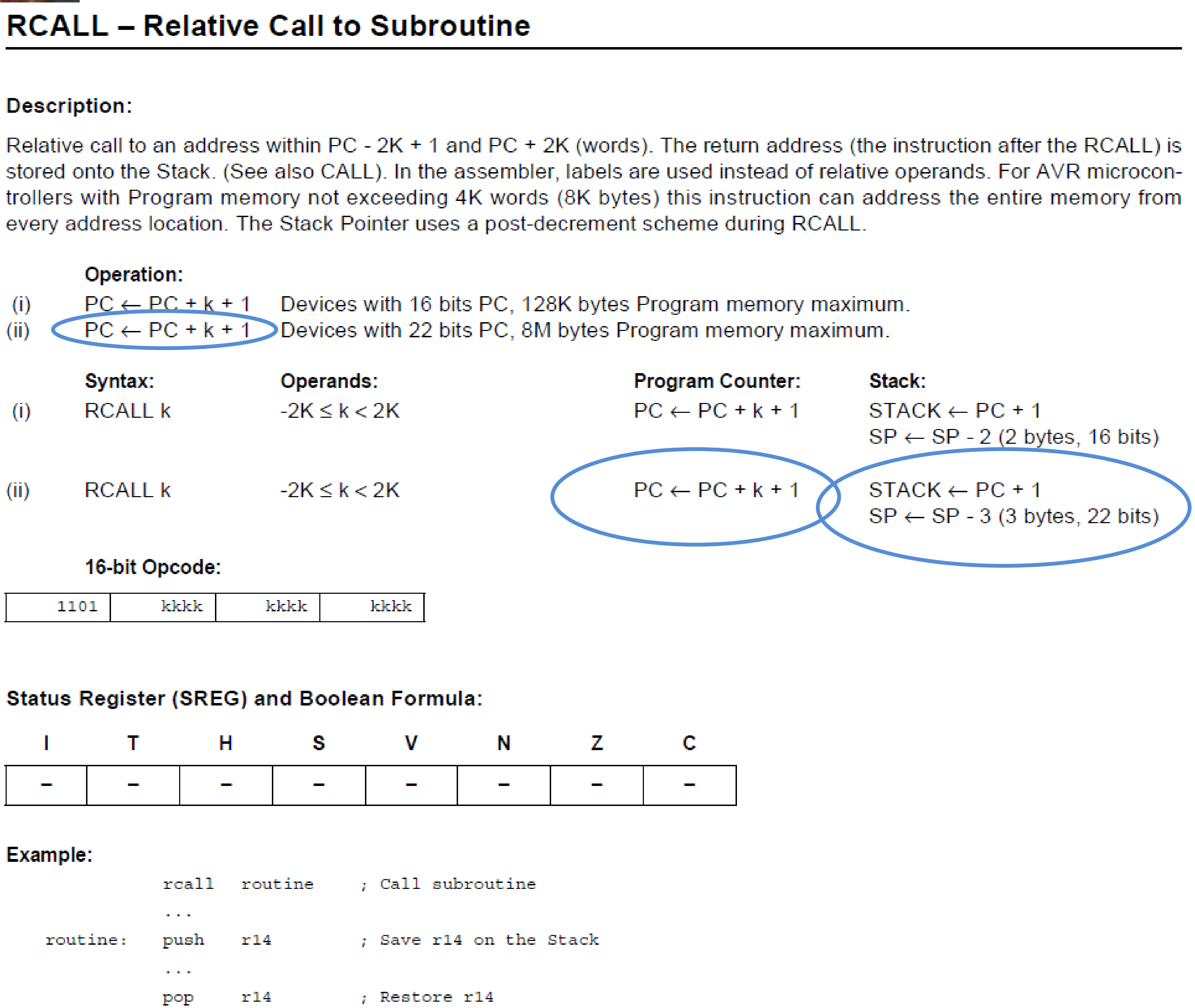


C:\Document\__Kurser\D2_E2\Datorteknik\HT_2015\Program_exam

```
ldi r16, 0xFF               ; Set Data Direction Regis
out DDRB, r16               ; port B as outputs
out DDRD, r16               ; port D as outputs

clr r16
out DDRA, r16               ; port A as inputs


in r16, PINA                ; read the content of port
com r16                     ; invert all bits in r16
andi r16, 0b00000010
cpi r16, 0b00000010
lsr r16

ldi r16, 1
ldi r17, 2
start_loop:

out PORTB, r16
out PORTD, r16              ; write the content of r1
mul r16, r17
mov r16, r0
rcall delay
rjmp start_loop

delay:
push r16                    ; save registers on stack

ldi r16,20

label_1:
rcall delay_x               ; call suboutine x ms
dec r16                     ; decrease counter
brne label_1                ; branch if not zero
```

Processor

Name	Value
Program Cou...	0x000014
Stack Pointer	0x21FF
X pointer	0x000000
Y pointer	0x000000
Z pointer	0x000000
Cycle Counter	21
Frequency	4.0000 MHz
Stop Watch	5.25 us
SREG	□I □T □H □S □V □N □Z □C
Registers	
R00	0x02
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x02



RCALL – Relative Call to Subroutine

Description:

Relative call to an address within PC - 2K + 1 and PC + 2K (words). The return address (the instruction after the RCALL) is stored onto the Stack. (See also CALL). In the assembler, labels are used instead of relative operands. For AVR microcontrollers with Program memory not exceeding 4K words (8K bytes) this instruction can address the entire memory from every address location. The Stack Pointer uses a post-decrement scheme during RCALL.

	Operation:	
(i)	PC ← PC + k + 1	Devices with 16 bits PC, 128K bytes Program memory maximum.
(ii)	PC ← PC + k + 1	Devices with 22 bits PC, 8M bytes Program memory maximum.

	Syntax:	Operands:	Program Counter:	Stack:

(i)	RCALL k	-2K ≤ k < 2K	PC ← PC + k + 1	STACK ← PC + 1
				SP ← SP - 2 (2 bytes, 16 bits)

(ii)	RCALL k	-2K ≤ k < 2K	PC ← PC + k + 1	STACK ← PC + 1
				SP ← SP - 3 (3 bytes, 22 bits)

16-bit Opcode:

1101	kkkk	kkkk	kkkk

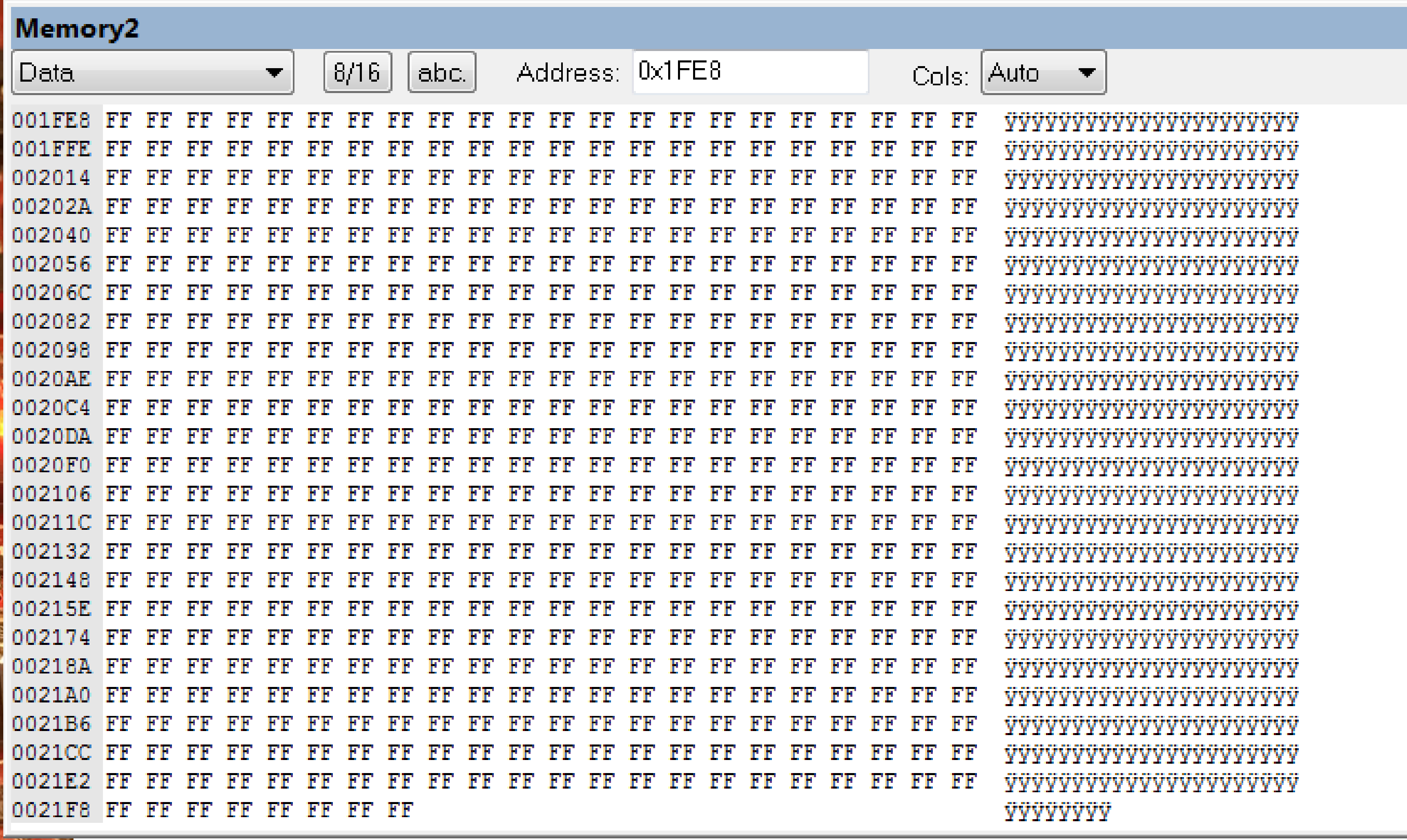
Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
            rcall   routine     ; Call subroutine
            ...
routine:    push    r14         ; Save r14 on the Stack
            ...
            pop     r14         ; Restore r14
```

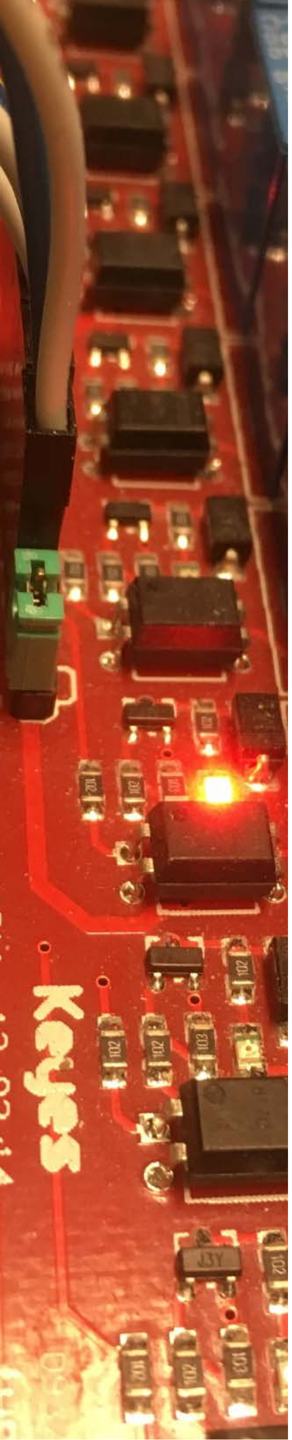
Data memory

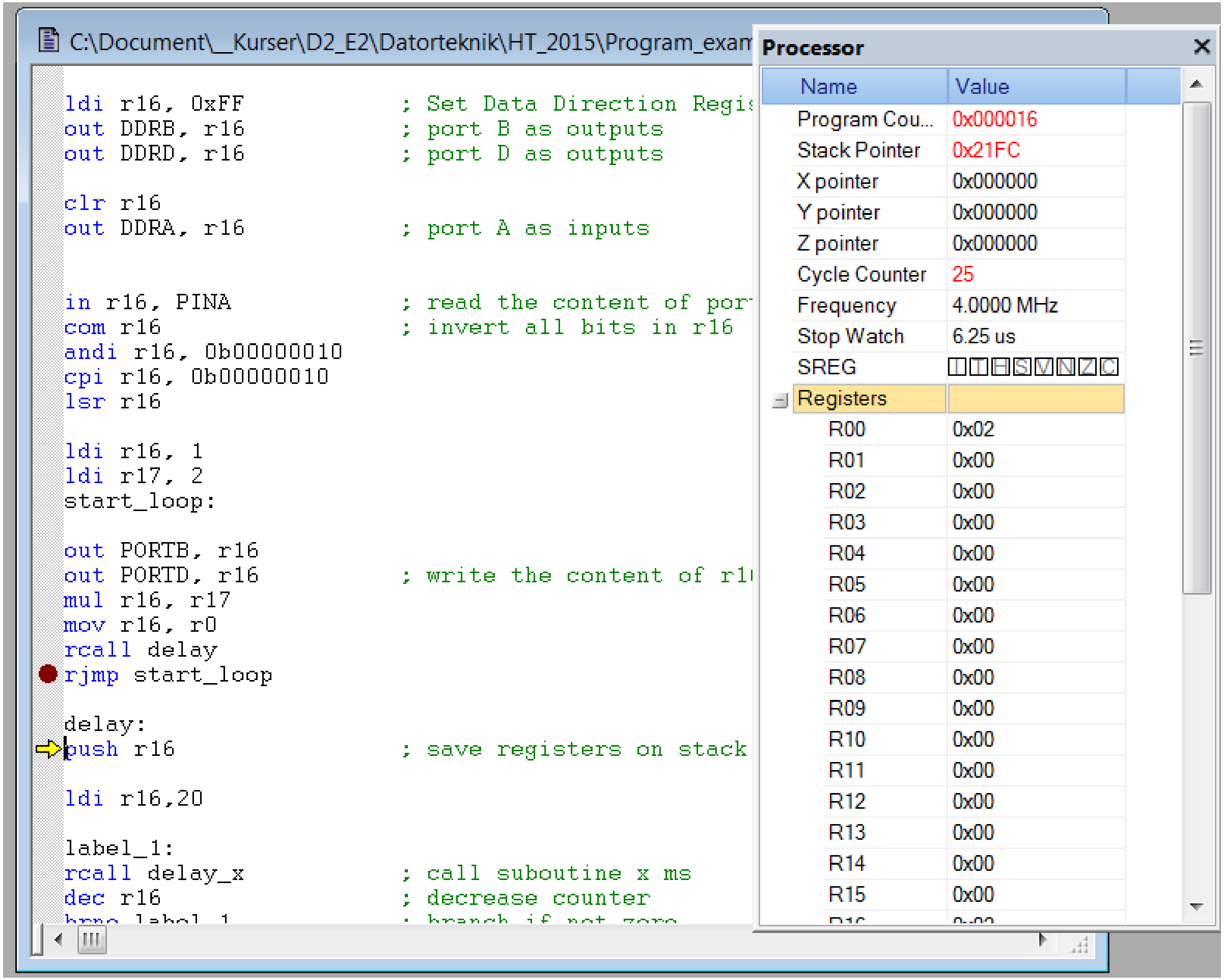


Memory2

Data ▾ | 8/16 | abc. | Address: 0x1FE8 | Cols: Auto ▾

```
001FE8 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
001FFE FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002014 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00202A FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002040 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002056 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00206C FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002082 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002098 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020AE FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020C4 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020DA FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020F0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002106 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00211C FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002132 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002148 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00215E FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002174 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00218A FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021A0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021B6 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021CC FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021E2 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021F8 FF FF FF FF FF FF FF FF                                            ÿÿÿÿÿÿÿÿ
```

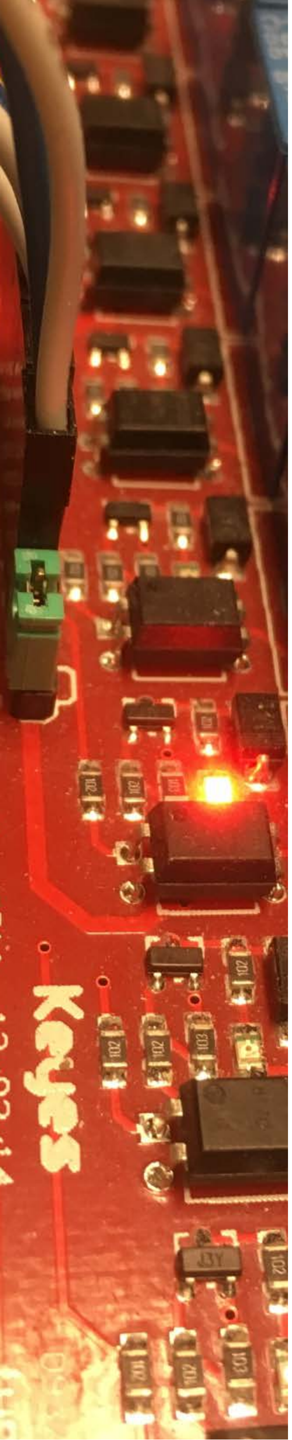




C:\Document\__Kurser\D2_E2\Datorteknik\HT_2015\Program_exam

```
ldi r16, 0xFF           ; Set Data Direction Regis
out DDRB, r16           ; port B as outputs
out DDRD, r16           ; port D as outputs

clr r16
out DDRA, r16           ; port A as inputs


in r16, PINA            ; read the content of por
com r16                 ; invert all bits in r16
andi r16, 0b00000010
cpi r16, 0b00000010
lsr r16

ldi r16, 1
ldi r17, 2
start_loop:

out PORTB, r16
out PORTD, r16          ; write the content of r1
mul r16, r17
mov r16, r0
rcall delay
rjmp start_loop

delay:
push r16                ; save registers on stack

ldi r16,20

label_1:
rcall delay_x           ; call suboutine x ms
dec r16                 ; decrease counter
brne label_1            ; branch if not zero
```

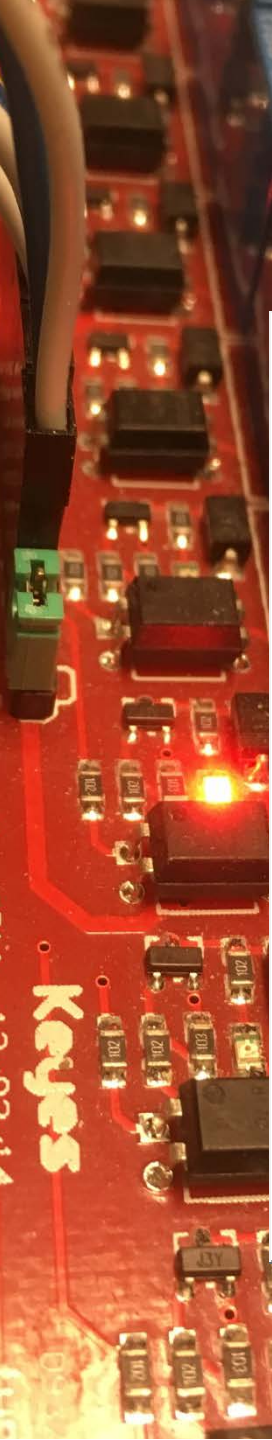
Name	Value
Program Cou...	0x000016
Stack Pointer	0x21FC
X pointer	0x000000
Y pointer	0x000000
Z pointer	0x000000
Cycle Counter	25
Frequency	4.0000 MHz
Stop Watch	6.25 us
SREG	I T H S V N Z C
Registers	
R00	0x02
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x02

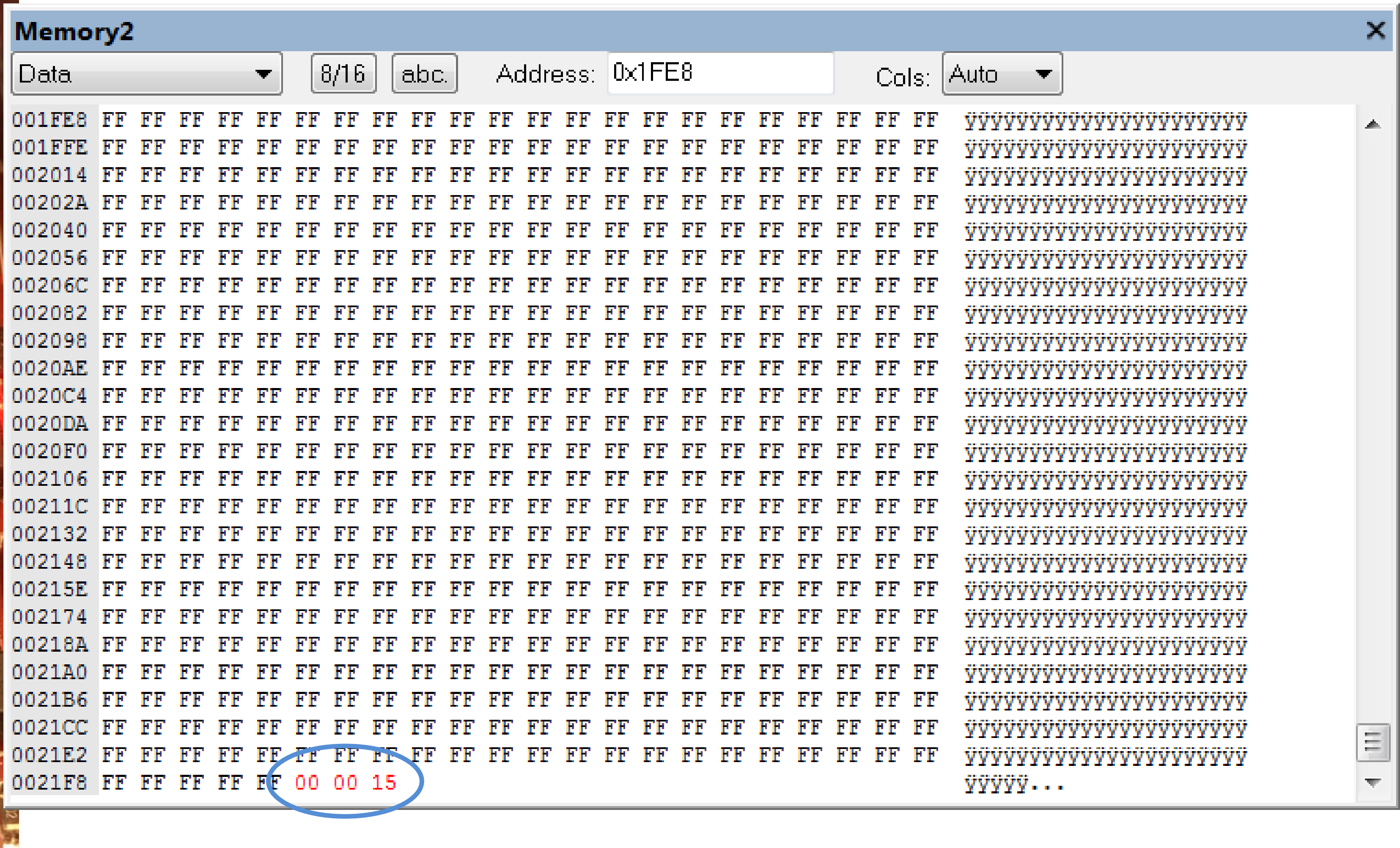


Question:

- What value will we find in the stack now?

Answer: The return address in program memory, which is 15.





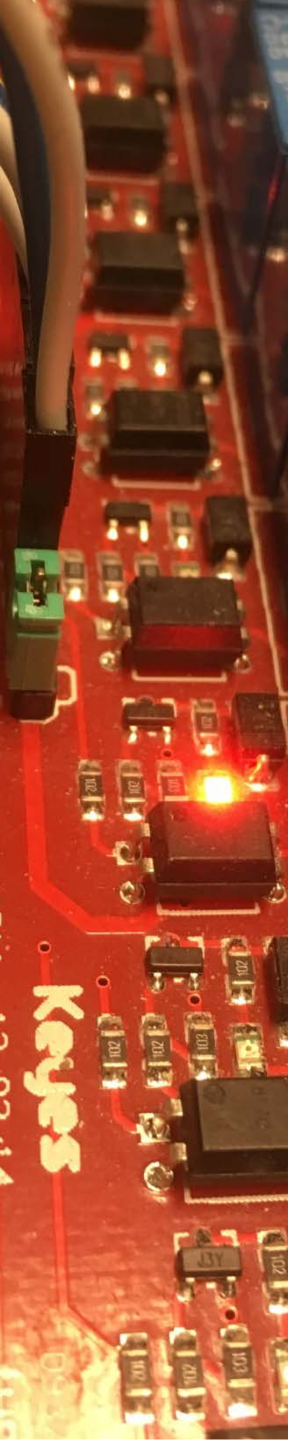
Memory2

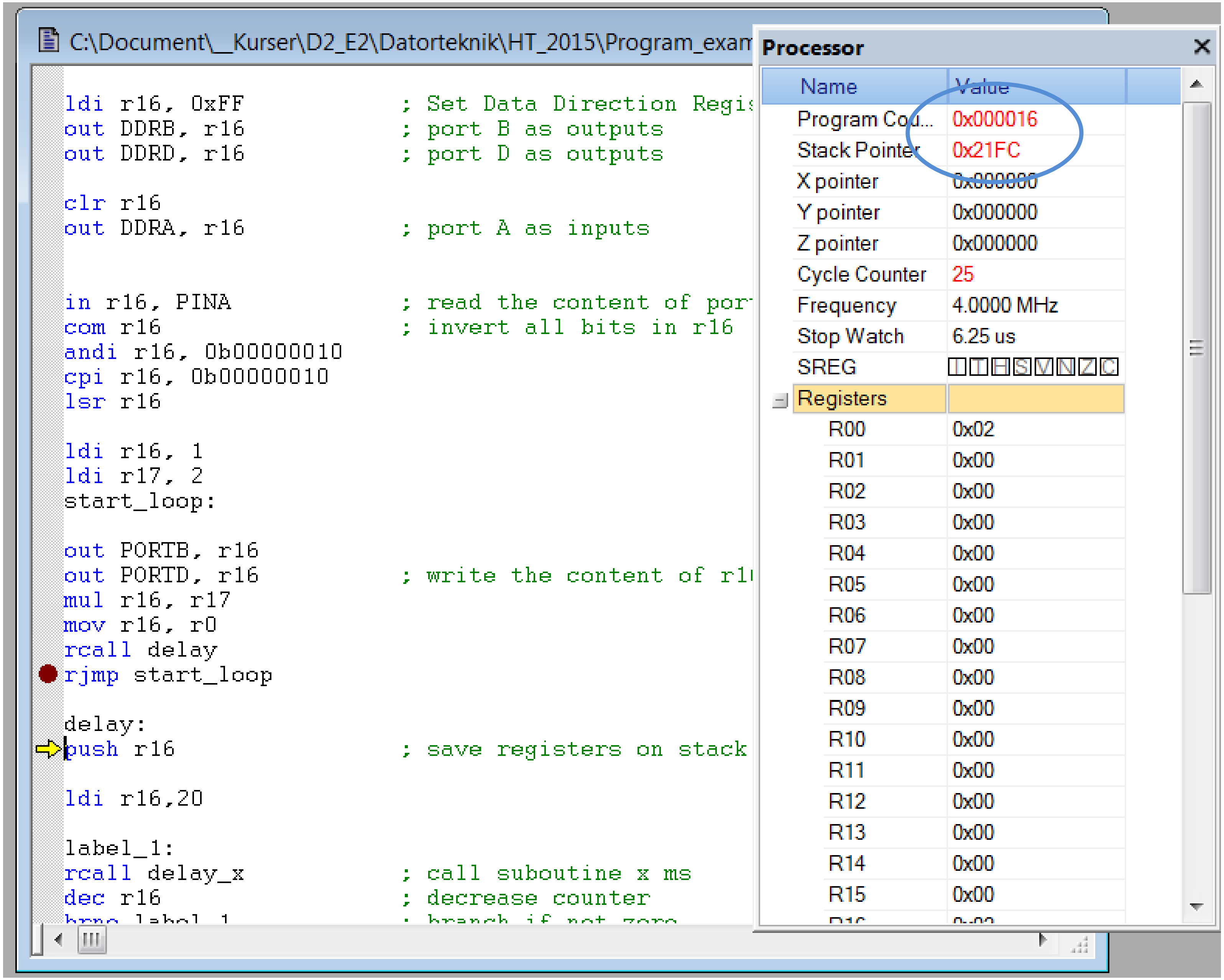
Data

Address: 0x1FE8

Cols: Auto

```
001FE8 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
001FFE FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002014 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00202A FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002040 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002056 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00206C FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002082 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002098 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020AE FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020C4 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020DA FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020F0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002106 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00211C FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002132 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002148 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00215E FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002174 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00218A FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021A0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021B6 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021CC FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021E2 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021F8 FF FF FF FF FF 00 00 15                                            ÿÿÿÿÿ...
```



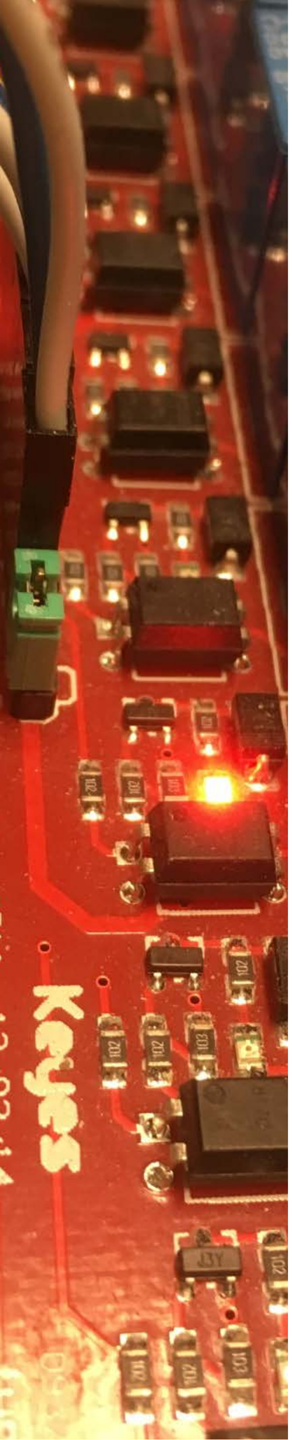


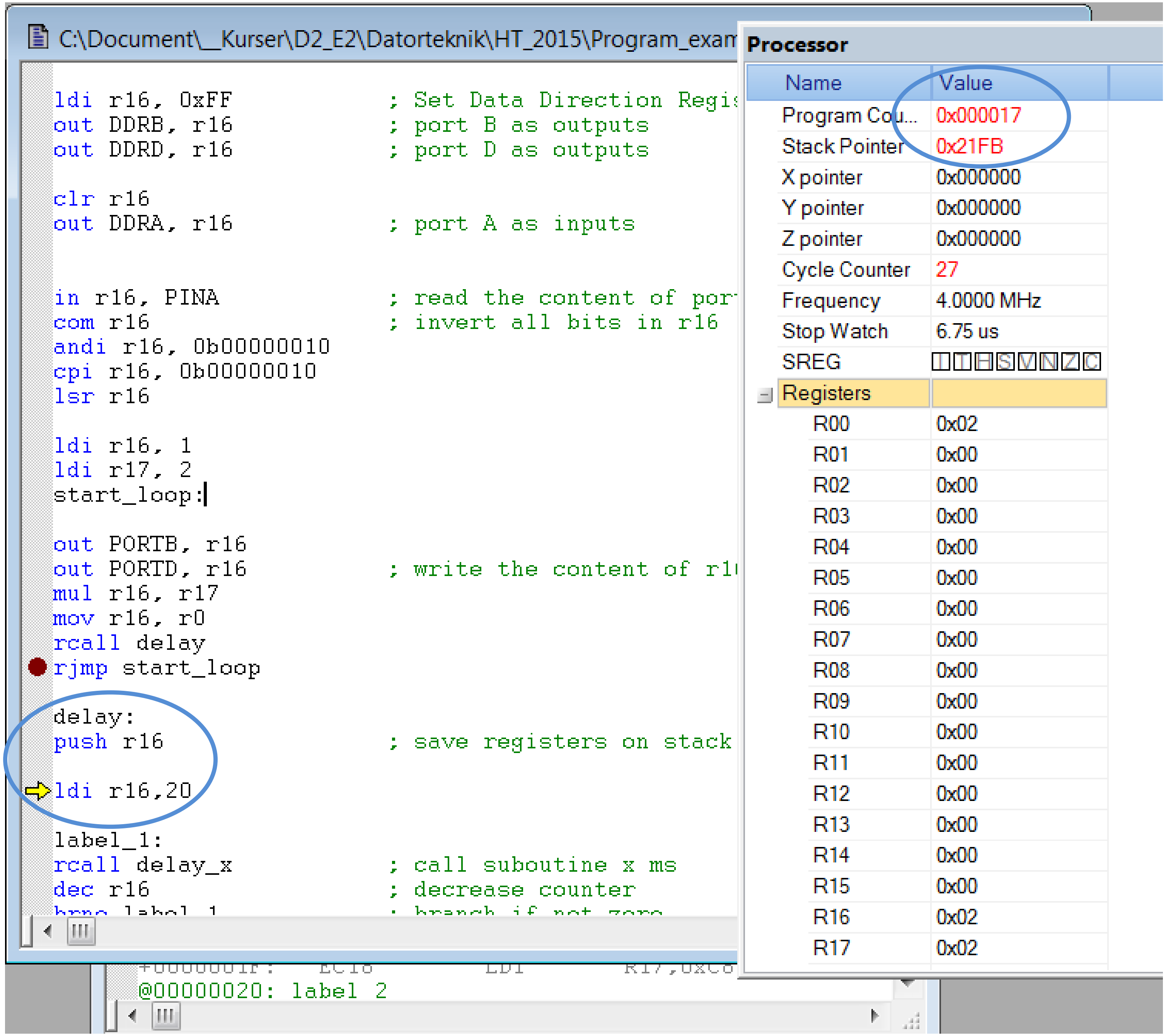
C:\Document\__Kurser\D2_E2\Datorteknik\HT_2015\Program_exam

```
ldi r16, 0xFF           ; Set Data Direction Regis
out DDRB, r16           ; port B as outputs
out DDRD, r16           ; port D as outputs

clr r16
out DDRA, r16           ; port A as inputs


in r16, PINA            ; read the content of por
com r16                 ; invert all bits in r16
andi r16, 0b00000010
cpi r16, 0b00000010
lsr r16

ldi r16, 1
ldi r17, 2
start_loop:

out PORTB, r16
out PORTD, r16          ; write the content of r1
mul r16, r17
mov r16, r0
rcall delay
rjmp start_loop

delay:
push r16                ; save registers on stack

ldi r16,20

label_1:
rcall delay_x           ; call suboutine x ms
dec r16                 ; decrease counter
brne label_1            ; branch if not zero
```

Processor

Name	Value
Program Cou...	0x000016
Stack Pointer	0x21FC
X pointer	0x000000
Y pointer	0x000000
Z pointer	0x000000
Cycle Counter	25
Frequency	4.0000 MHz
Stop Watch	6.25 us
SREG	I T H S V N Z C
Registers	
R00	0x02
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x02





C:\Document\__Kurser\D2_E2\Datorteknik\HT_2015\Program_exam

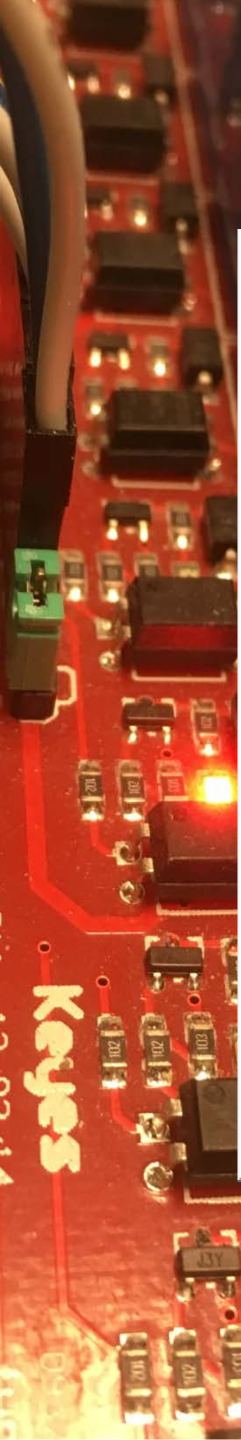
```
ldi r16, 0xFF             ; Set Data Direction Regis
out DDRB, r16             ; port B as outputs
out DDRD, r16             ; port D as outputs

clr r16
out DDRA, r16             ; port A as inputs


in r16, PINA              ; read the content of por
com r16                   ; invert all bits in r16
andi r16, 0b00000010
cpi r16, 0b00000010
lsr r16

ldi r16, 1
ldi r17, 2
start_loop:

out PORTB, r16
out PORTD, r16            ; write the content of r1
mul r16, r17
mov r16, r0
rcall delay
rjmp start_loop

delay:
push r16                  ; save registers on stack

ldi r16,20

label_1:
rcall delay_x             ; call suboutine x ms
dec r16                   ; decrease counter
brne label_1              ; branch if not zero
```

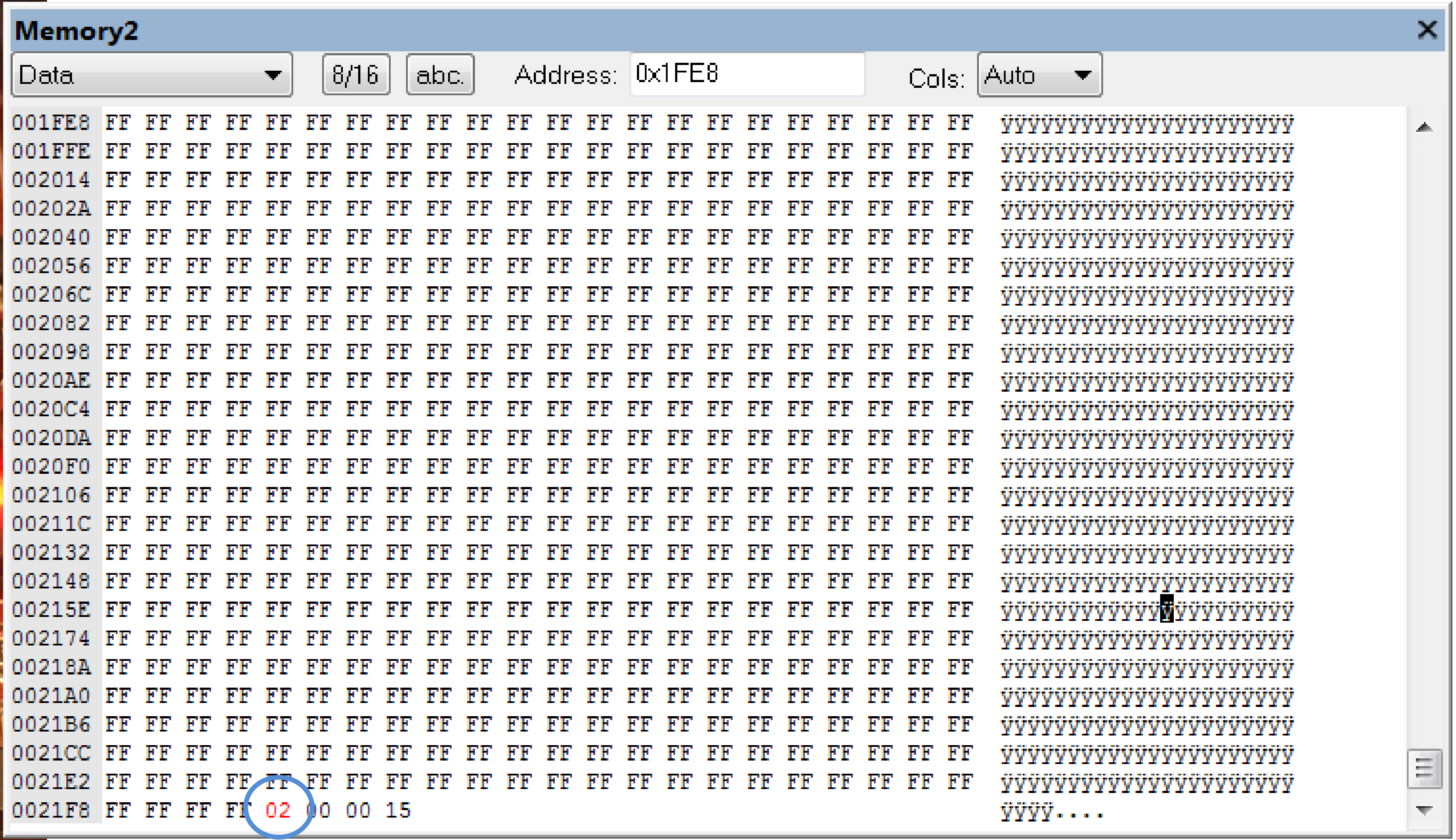
```
+0000001F:    EC18        LDI        R17,0xC8
@00000020: label 2
```

Processor

Name	Value
Program Cou...	0x000017
Stack Pointer	0x21FB
X pointer	0x000000
Y pointer	0x000000
Z pointer	0x000000
Cycle Counter	27
Frequency	4.0000 MHz
Stop Watch	6.75 us
SREG	I T H S V N Z C
Registers	
R00	0x02
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x02
R17	0x02

The Stack



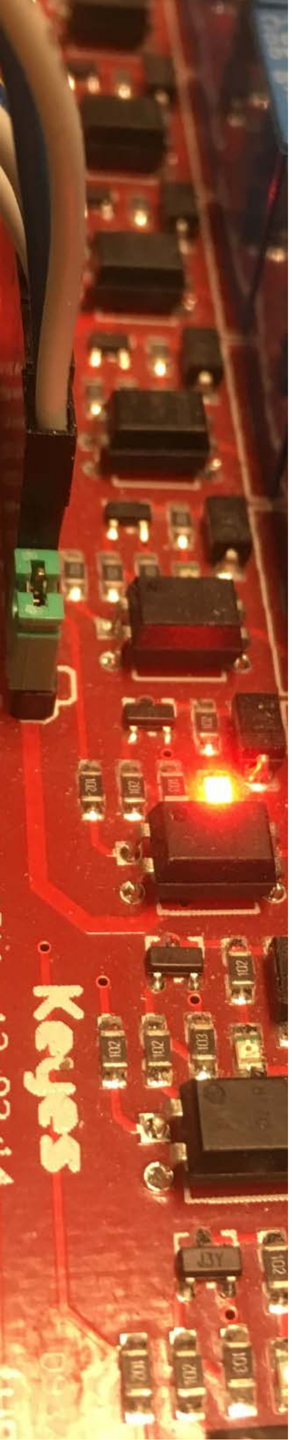


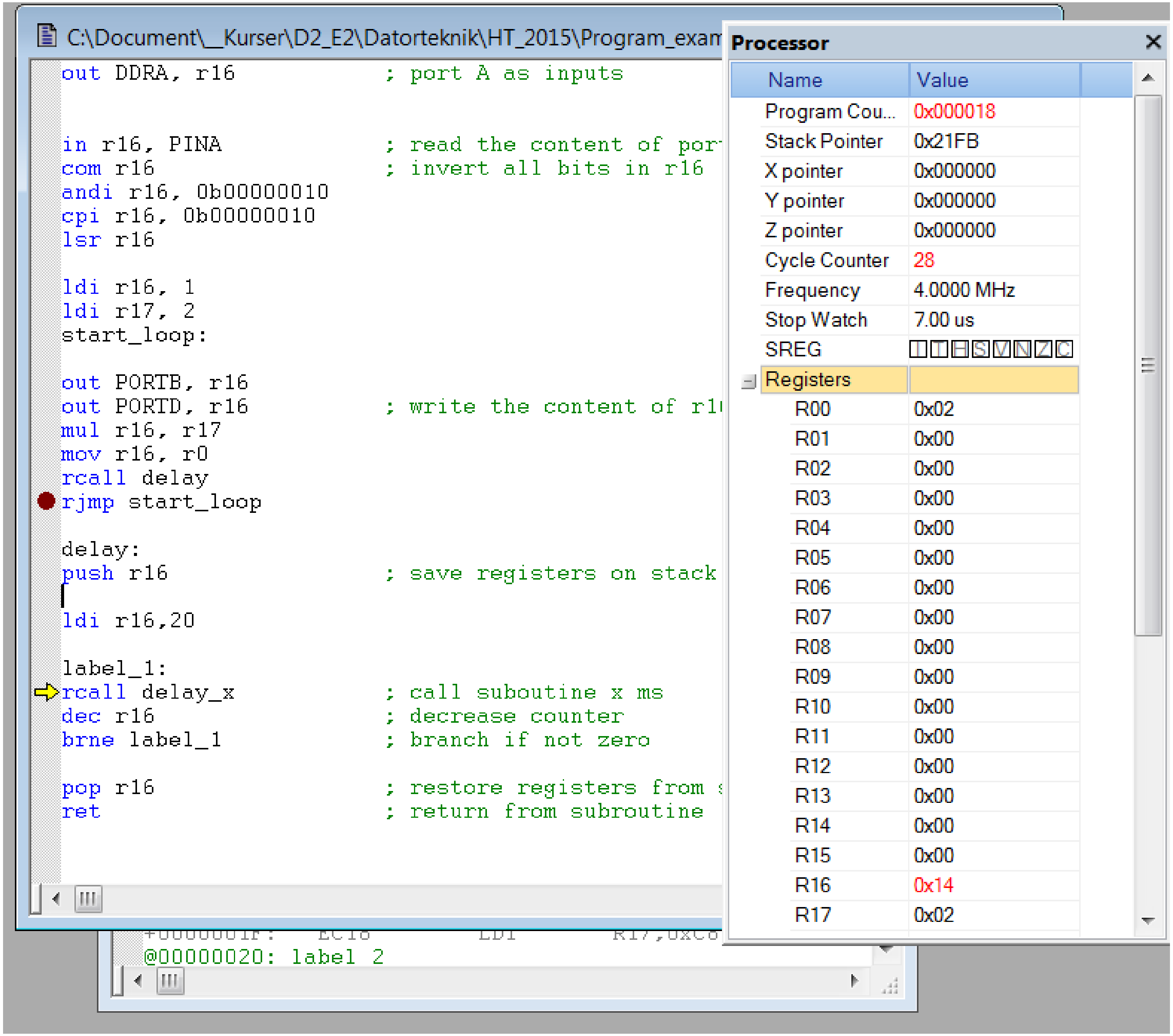
Data

Address: 0x1FE8

Cols: Auto

```
001FE8 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
001FFE FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002014 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00202A FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002040 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002056 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00206C FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002082 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002098 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020AE FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020C4 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020DA FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020F0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002106 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00211C FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002132 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002148 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00215E FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002174 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00218A FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021A0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021B6 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021CC FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021E2 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021F8 FF FF FF FF 02 00 00 15                                            ÿÿÿÿ....
```





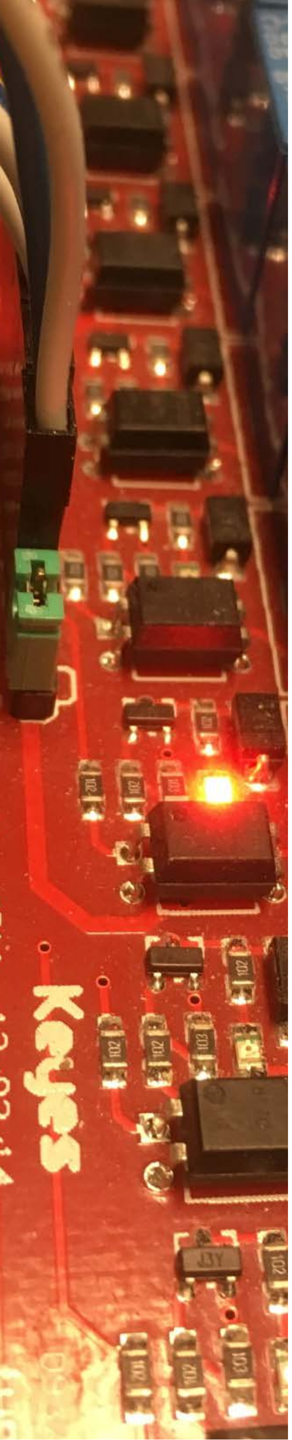
C:\Document\__Kurser\D2_E2\Datorteknik\HT_2015\Program_exam

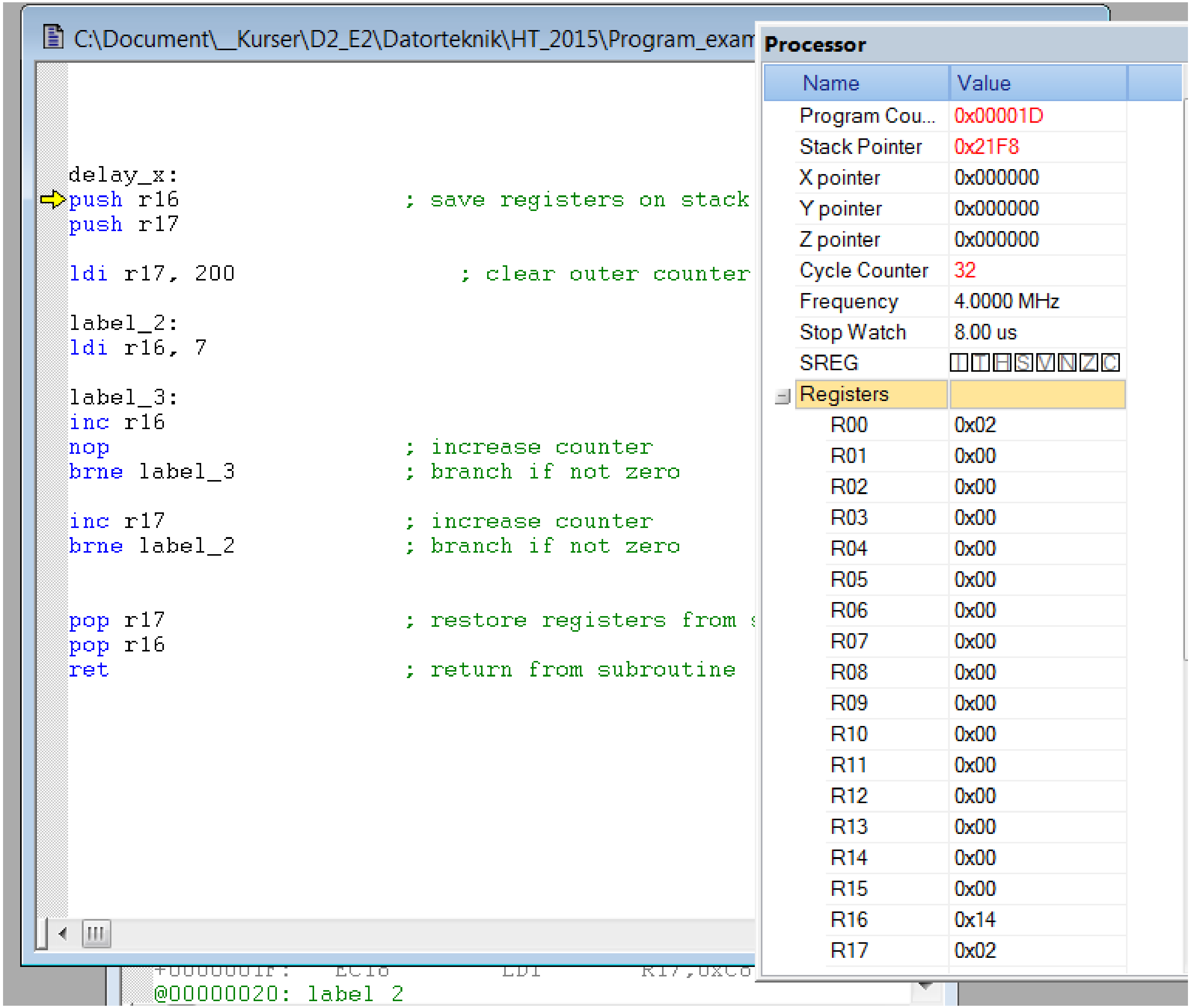
```
out DDRA, r16              ; port A as inputs


in r16, PINA               ; read the content of por
com r16                    ; invert all bits in r16
andi r16, 0b00000010
cpi r16, 0b00000010
lsr r16

ldi r16, 1
ldi r17, 2
start_loop:

out PORTB, r16
out PORTD, r16             ; write the content of r1
mul r16, r17
mov r16, r0
rcall delay
rjmp start_loop

delay:
push r16                   ; save registers on stack

ldi r16,20

label_1:
rcall delay_x              ; call suboutine x ms
dec r16                    ; decrease counter
brne label_1               ; branch if not zero

pop r16                    ; restore registers from s
ret                        ; return from subroutine
```

```
@00000020: label 2
```

Processor

Name	Value
Program Cou...	0x000018
Stack Pointer	0x21FB
X pointer	0x000000
Y pointer	0x000000
Z pointer	0x000000
Cycle Counter	28
Frequency	4.0000 MHz
Stop Watch	7.00 us
SREG	ITHSVNZC
Registers	
R00	0x02
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x14
R17	0x02





C:\Document\__Kurser\D2_E2\Datorteknik\HT_2015\Program_exam

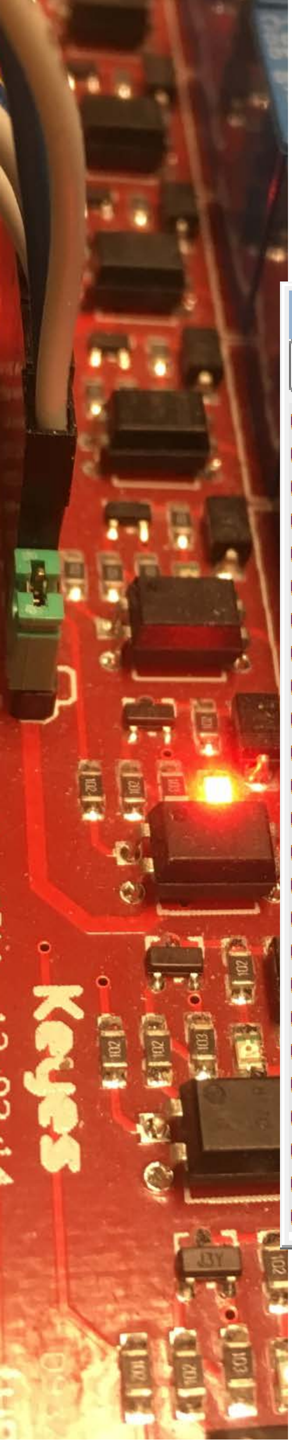
```
delay_x:
push r16                ; save registers on stack
push r17

ldi r17, 200                ; clear outer counter

label_2:
ldi r16, 7

label_3:
inc r16
nop                     ; increase counter
brne label_3            ; branch if not zero

inc r17                 ; increase counter
brne label_2            ; branch if not zero


pop r17                 ; restore registers from s
pop r16
ret                     ; return from subroutine
```

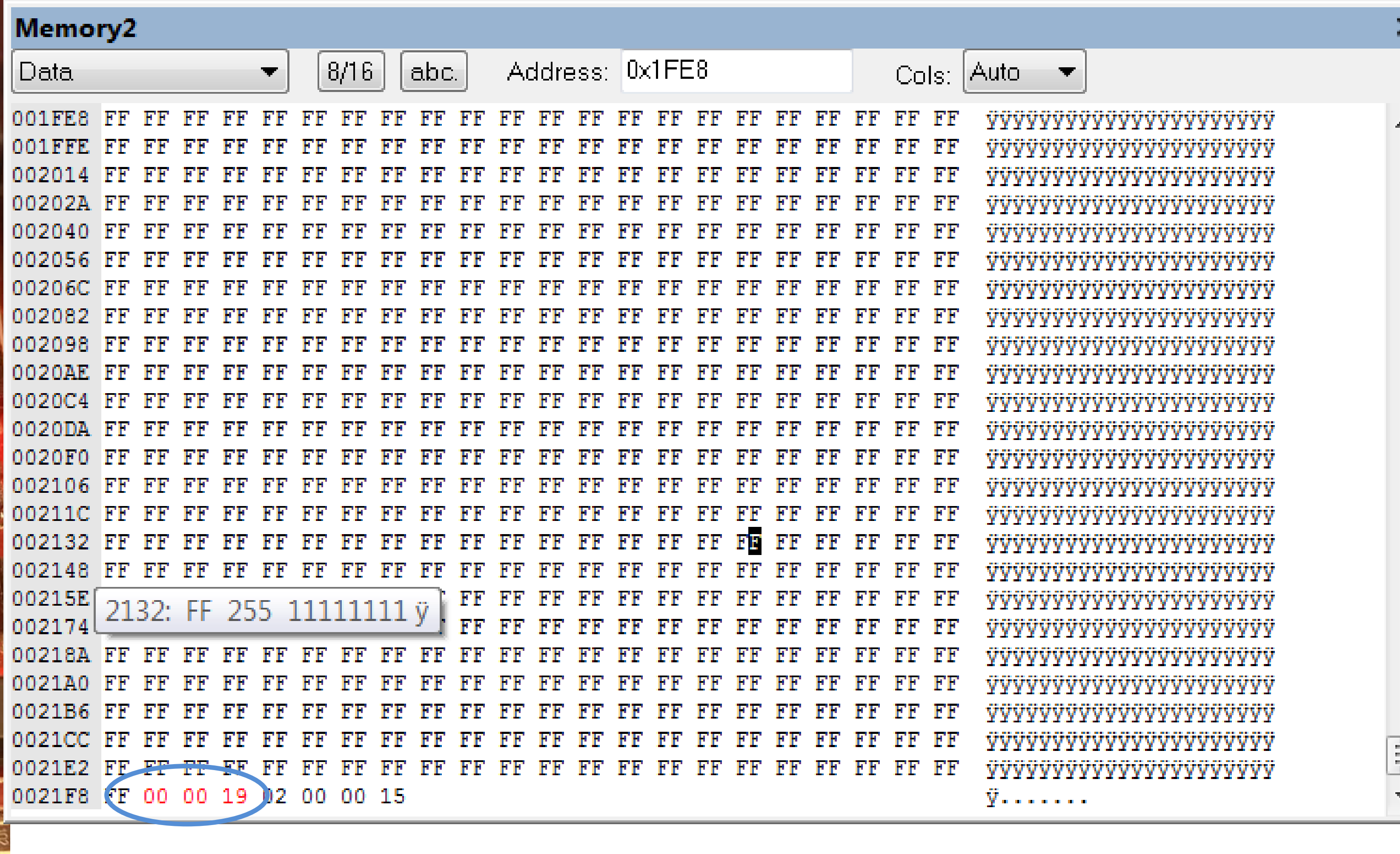
```
+0000001F:   EC16        LDI       R17,0xC8
@00000020: label_2
```

Processor

Name	Value
Program Cou...	0x00001D
Stack Pointer	0x21F8
X pointer	0x000000
Y pointer	0x000000
Z pointer	0x000000
Cycle Counter	32
Frequency	4.0000 MHz
Stop Watch	8.00 us
SREG	I T H S V N Z C
Registers	
R00	0x02
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x14
R17	0x02

The Stack



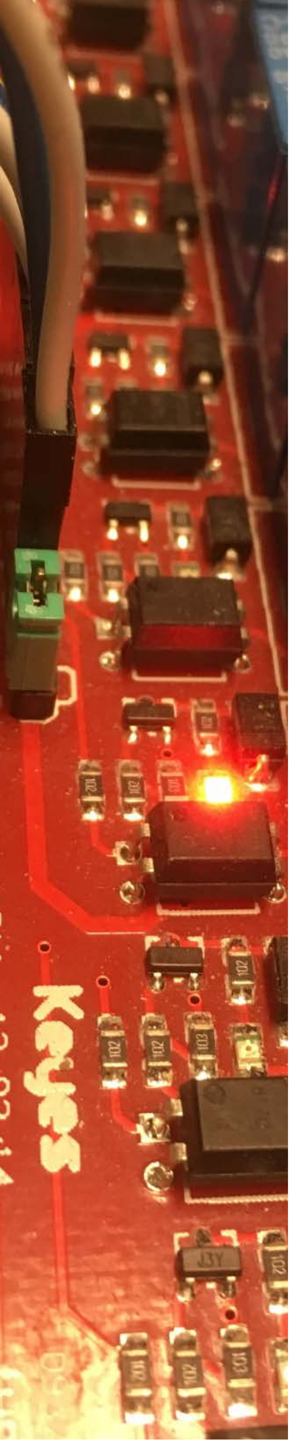


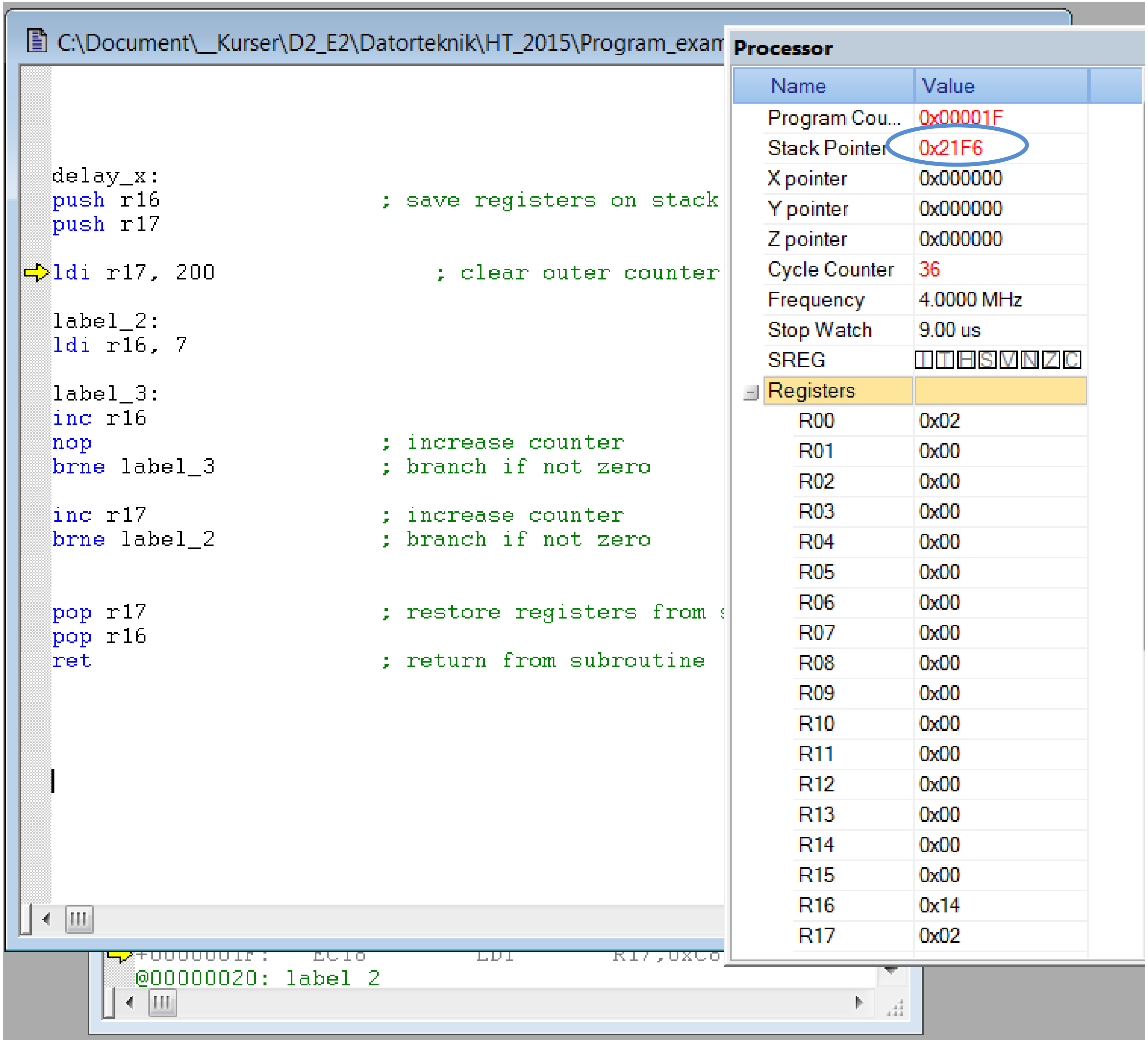
Memory2

Data ▼ | 8/16 | abc. | Address: 0x1FE8 | Cols: Auto ▼

```
001FE8 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
001FFE FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002014 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00202A FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002040 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002056 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00206C FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002082 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002098 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020AE FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020C4 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020DA FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020F0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002106 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00211C FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002132 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002148 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00215E                            FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002174                            FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00218A FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021A0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021B6 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021CC FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021E2 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021F8 FF 00 00 19 02 00 00 15                                            ÿ.......
```

2132: FF 255 11111111 ÿ





C:\Document\__Kurser\D2_E2\Datorteknik\HT_2015\Program_exam

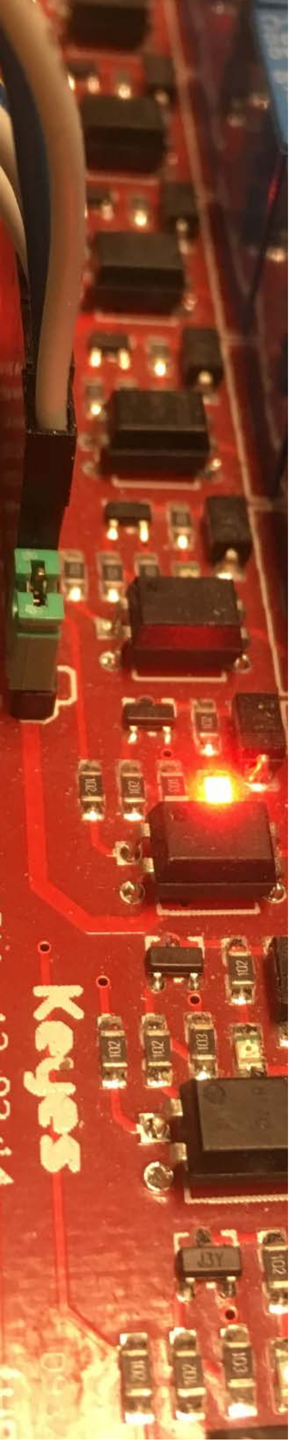
```
delay_x:
push r16                  ; save registers on stack
push r17

ldi r17, 200                  ; clear outer counter

label_2:
ldi r16, 7

label_3:
inc r16
nop                       ; increase counter
brne label_3              ; branch if not zero

inc r17                   ; increase counter
brne label_2              ; branch if not zero


pop r17                   ; restore registers from s
pop r16
ret                       ; return from subroutine
```

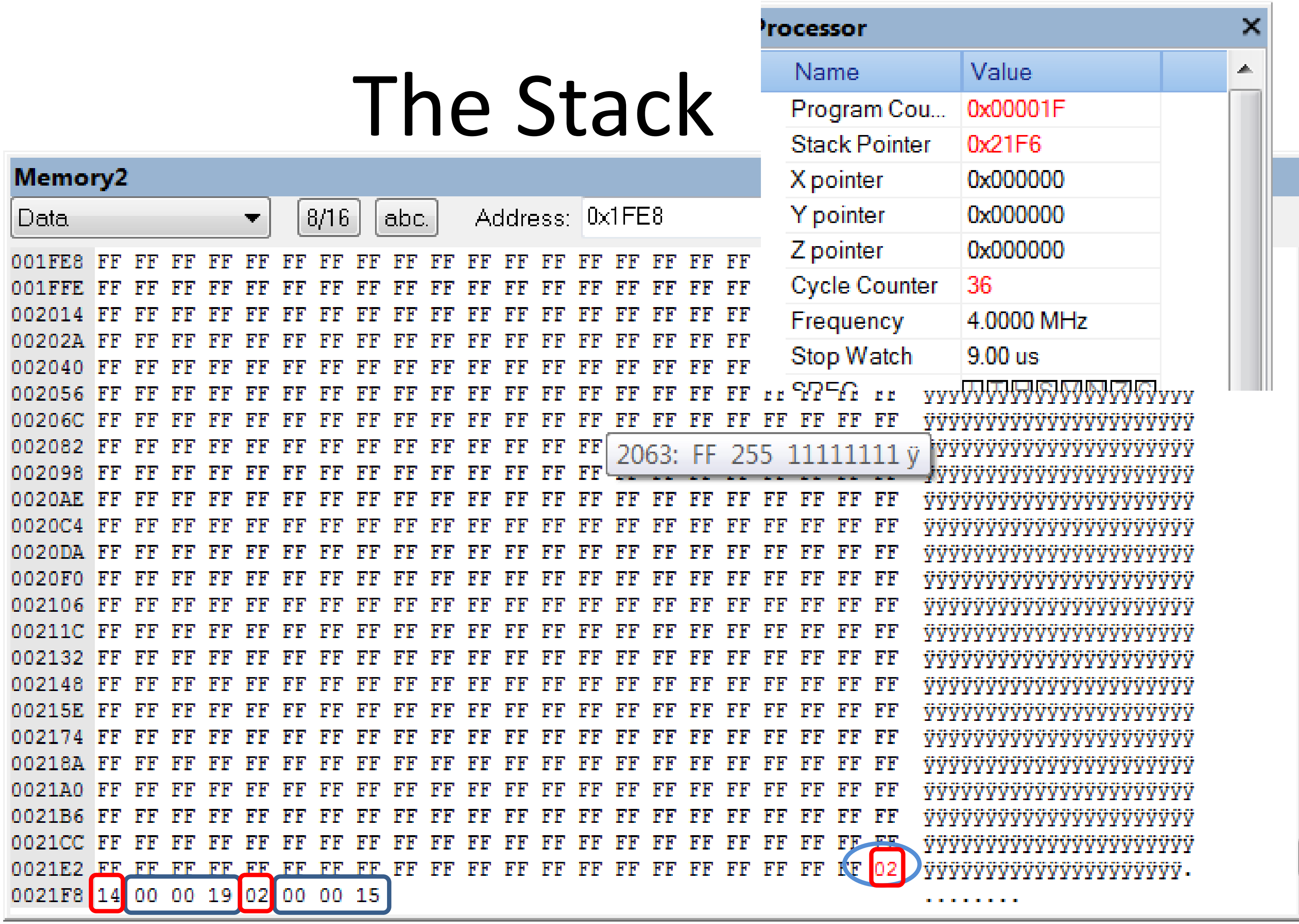
```
+0000001F:   EC18        LDI       R17,0xC8
@00000020: label_2
```

Processor

Name	Value
Program Cou...	0x00001F
Stack Pointer	0x21F6
X pointer	0x000000
Y pointer	0x000000
Z pointer	0x000000
Cycle Counter	36
Frequency	4.0000 MHz
Stop Watch	9.00 us
SREG	I T H S V N Z C
Registers	
R00	0x02
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x14
R17	0x02

The Stack



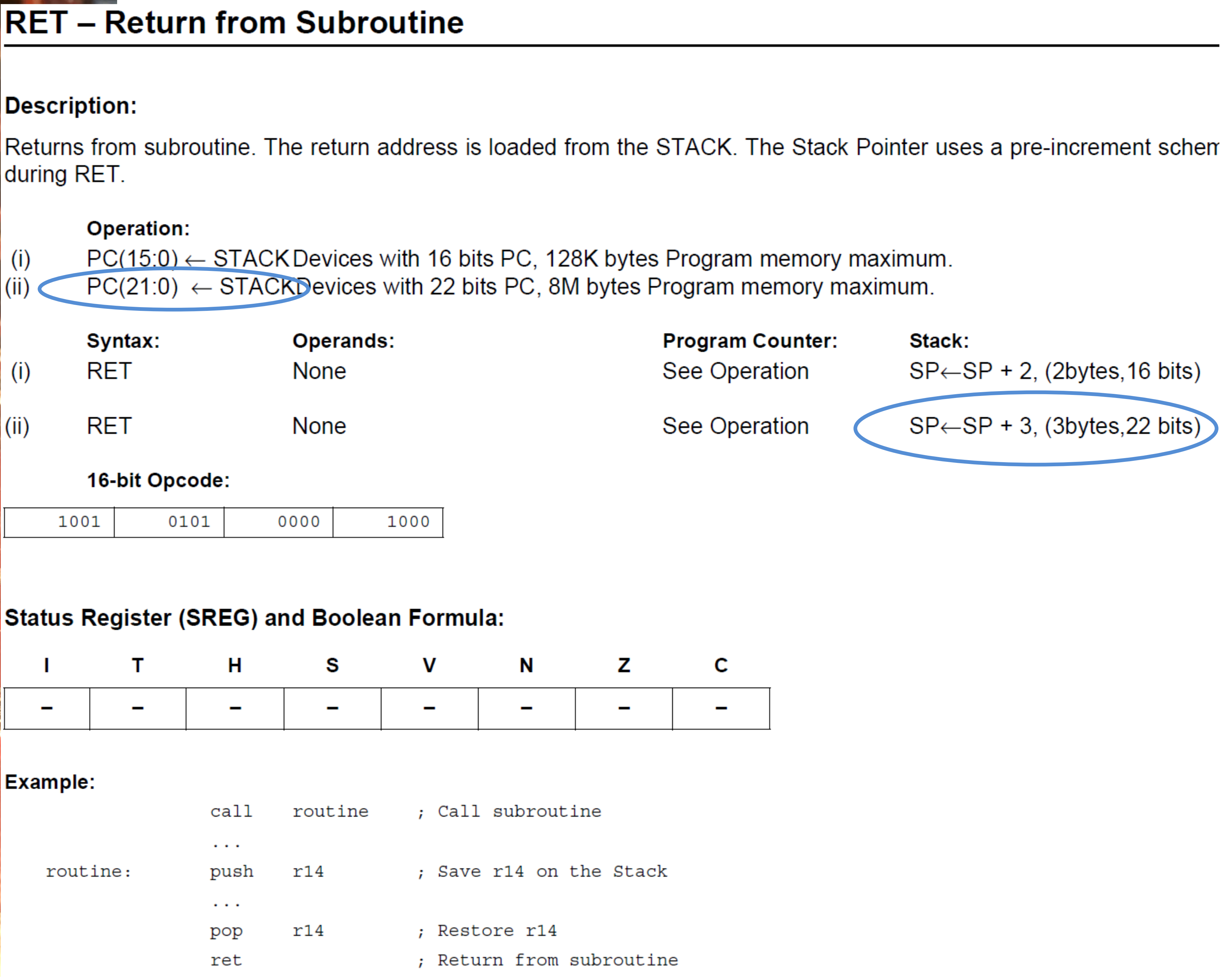


Address: 0x1FE8

```
001FE8 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
001FFE FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
002014 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00202A FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
002040 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
002056 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00206C FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002082 FF FF FF FF FF FF FF FF FF FF FF FF FF FF                          ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002098 FF FF FF FF FF FF FF FF FF FF FF FF FF FF                          ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020AE FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020C4 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020DA FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020F0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002106 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00211C FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002132 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002148 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00215E FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002174 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00218A FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021A0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021B6 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021CC FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021E2 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 02  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ.
0021F8 14 00 00 19 02 00 00 15                                            ........
```

```
2063: FF 255 11111111 ÿ
```

Name	Value
Program Cou...	0x00001F
Stack Pointer	0x21F6
X pointer	0x000000
Y pointer	0x000000
Z pointer	0x000000
Cycle Counter	36
Frequency	4.0000 MHz
Stop Watch	9.00 us
SREG	□□□□□□□□



RET – Return from Subroutine

Description:

Returns from subroutine. The return address is loaded from the STACK. The Stack Pointer uses a pre-increment schem during RET.

Operation:

(i) PC(15:0) ← STACKDevices with 16 bits PC, 128K bytes Program memory maximum.

(ii) PC(21:0) ← STACKDevices with 22 bits PC, 8M bytes Program memory maximum.

	Syntax:	Operands:	Program Counter:	Stack:
(i)	RET	None	See Operation	SP←SP + 2, (2bytes,16 bits)
(ii)	RET	None	See Operation	SP←SP + 3, (3bytes,22 bits)

16-bit Opcode:

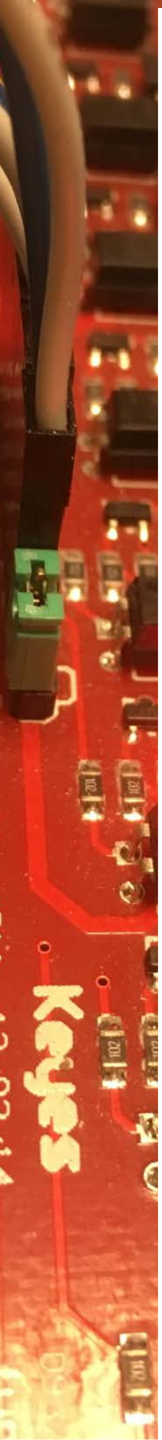
1001	0101	0000	1000

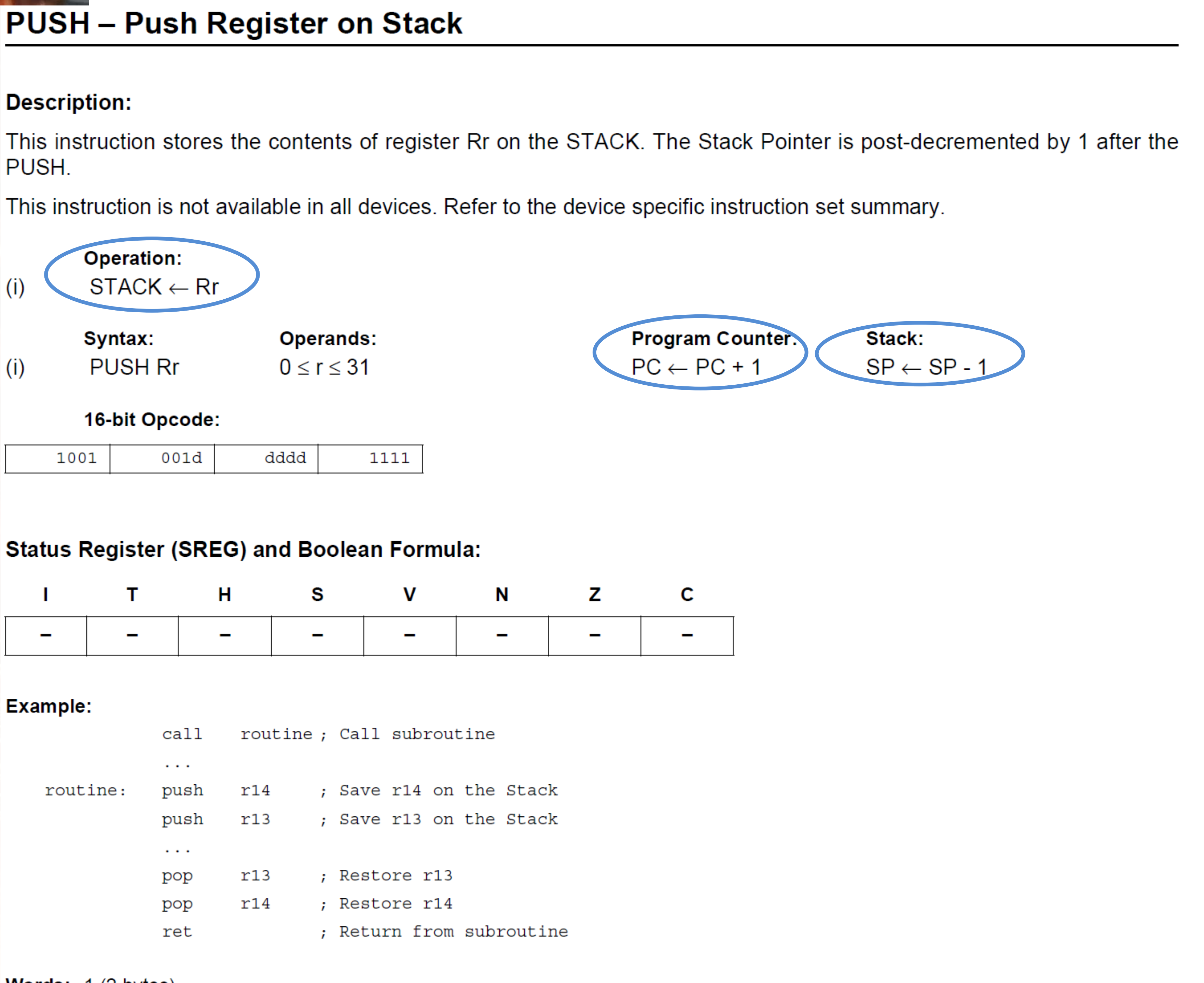
Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
                call    routine     ; Call subroutine
                ...
    routine:    push    r14         ; Save r14 on the Stack
                ...
                pop     r14         ; Restore r14
                ret                 ; Return from subroutine
```





PUSH – Push Register on Stack

Description:

This instruction stores the contents of register Rr on the STACK. The Stack Pointer is post-decremented by 1 after the PUSH.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) STACK ← Rr

	Syntax:	**Operands:**	**Program Counter:**	**Stack:**
(i)	PUSH Rr	0 ≤ r ≤ 31	PC ← PC + 1	SP ← SP - 1

16-bit Opcode:

1001	001d	dddd	1111

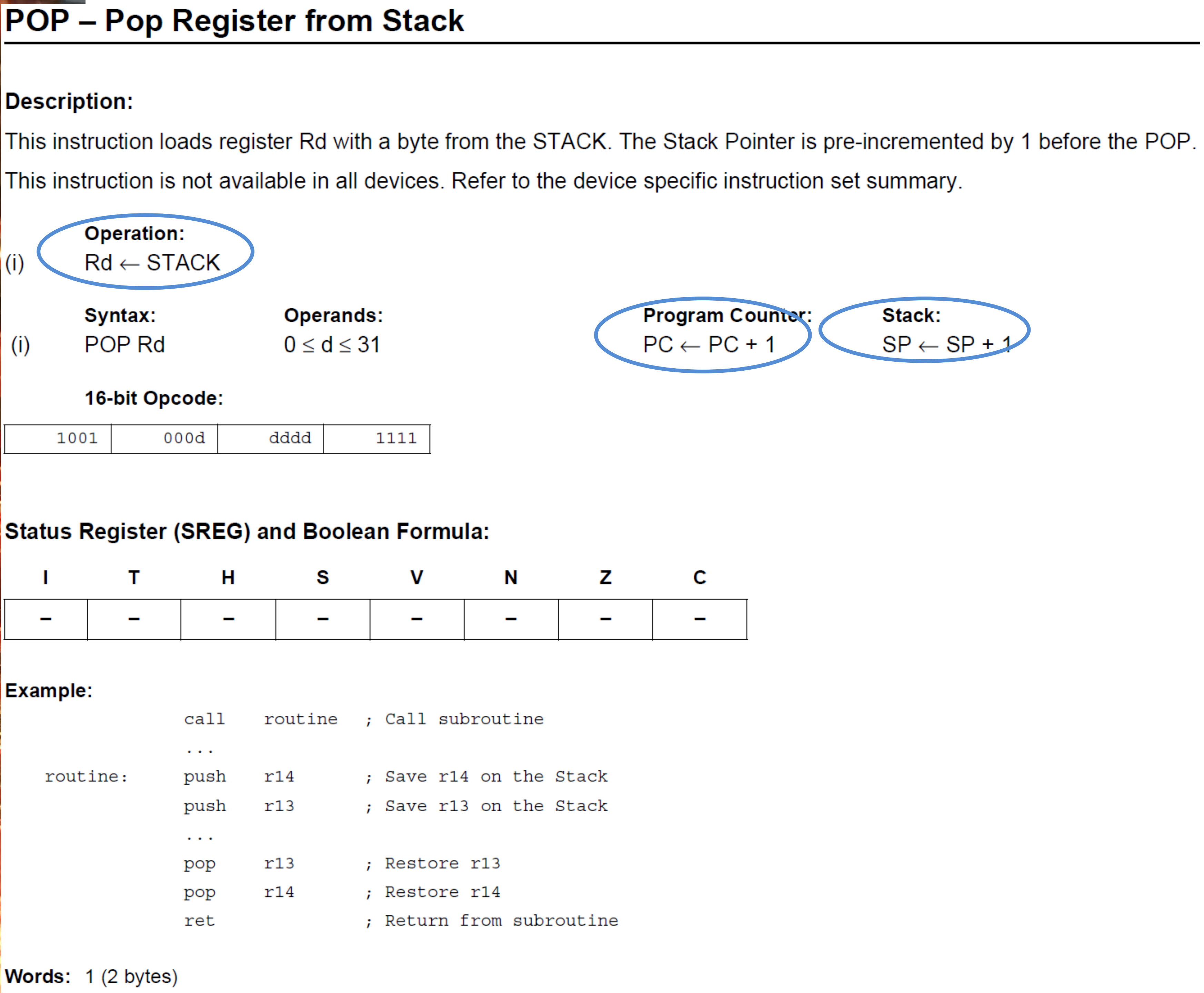
Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
             call    routine ; Call subroutine
             ...
   routine:  push    r14     ; Save r14 on the Stack
             push    r13     ; Save r13 on the Stack
             ...
             pop     r13     ; Restore r13
             pop     r14     ; Restore r14
             ret             ; Return from subroutine
```

Words: 1 (2 bytes)



POP – Pop Register from Stack

Description:

This instruction loads register Rd with a byte from the STACK. The Stack Pointer is pre-incremented by 1 before the POP.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

(i) **Operation:**
Rd ← STACK

	Syntax:	**Operands:**	**Program Counter:**	**Stack:**
(i)	POP Rd	0 ≤ d ≤ 31	PC ← PC + 1	SP ← SP + 1

16-bit Opcode:

1001	000d	dddd	1111

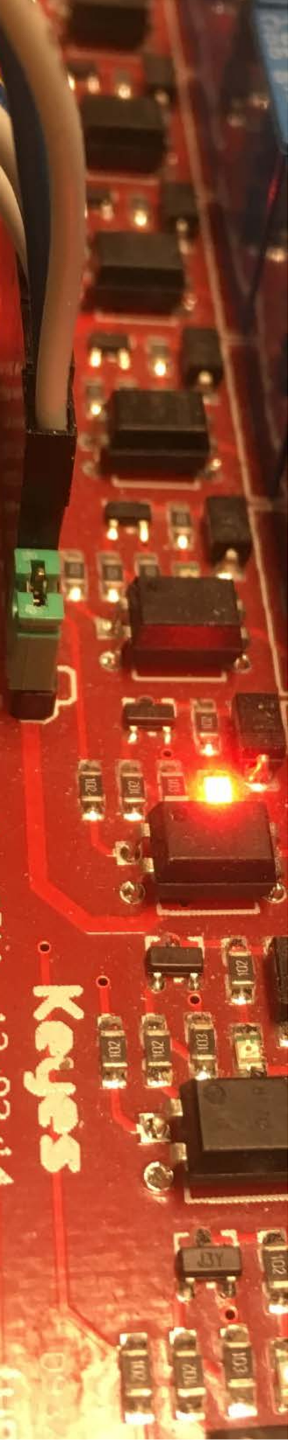
Status Register (SREG) and Boolean Formula:

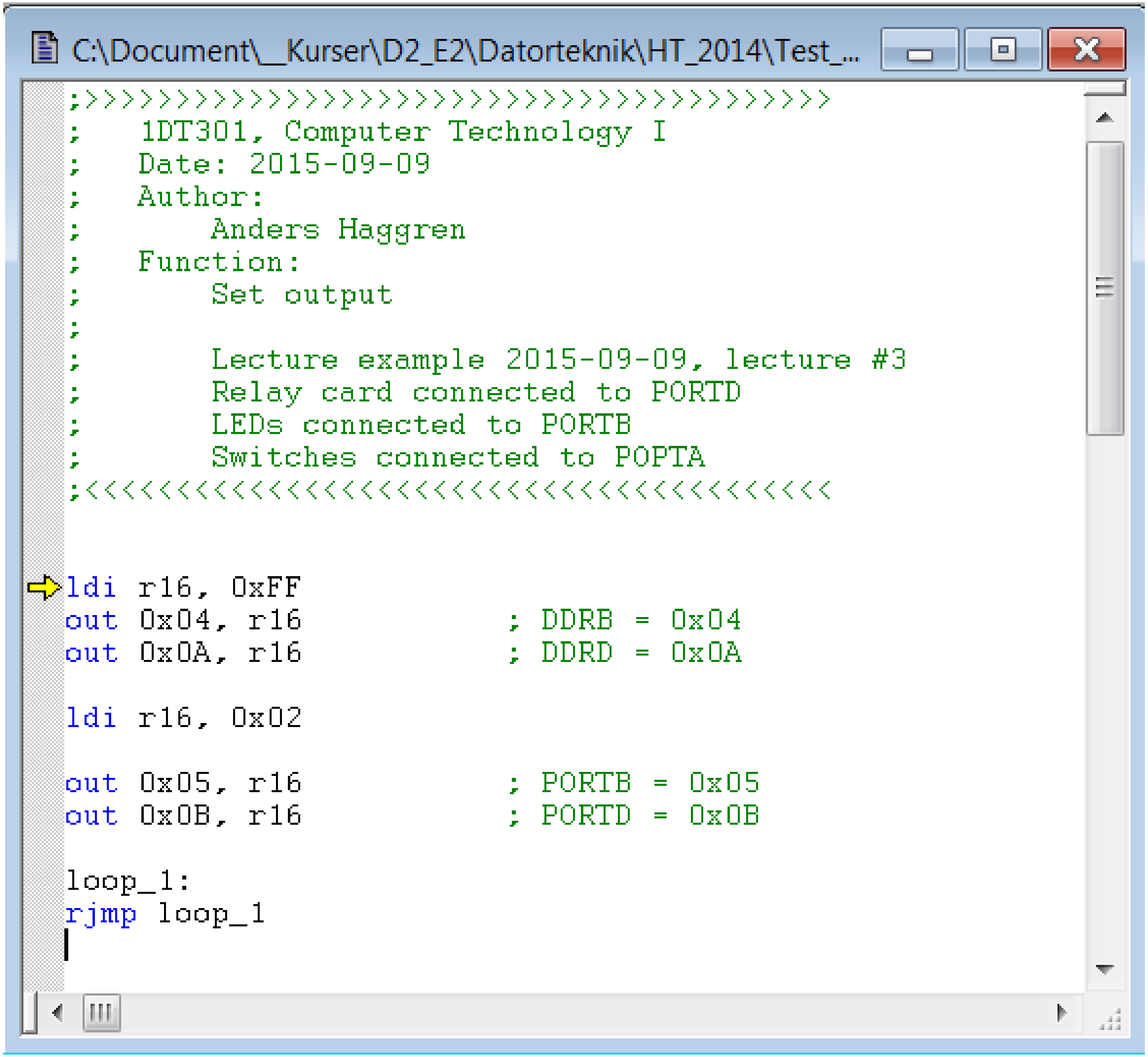
I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
                call    routine   ; Call subroutine
                ...
    routine:    push    r14       ; Save r14 on the Stack
                push    r13       ; Save r13 on the Stack
                ...
                pop     r13       ; Restore r13
                pop     r14       ; Restore r14
                ret               ; Return from subroutine
```

Words: 1 (2 bytes)





```
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;   1DT301, Computer Technology I
;   Date: 2015-09-09
;   Author:
;       Anders Haggren
;   Function:
;       Set output
;
;       Lecture example 2015-09-09, lecture #3
;       Relay card connected to PORTD
;       LEDs connected to PORTB
;       Switches connected to POPTA
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<


ldi r16, 0xFF
out 0x04, r16           ; DDRB = 0x04
out 0x0A, r16           ; DDRD = 0x0A

ldi r16, 0x02

out 0x05, r16           ; PORTB = 0x05
out 0x0B, r16           ; PORTD = 0x0B

loop_1:
rjmp loop_1
```

Section 4

AVR Assembler User Guide

4.1 Introduction

Welcome to the Atmel *AVR* Assembler. This manual describes the usage of the Assembler. The Assembler covers the whole range of microcontrollers in the AT90S family.

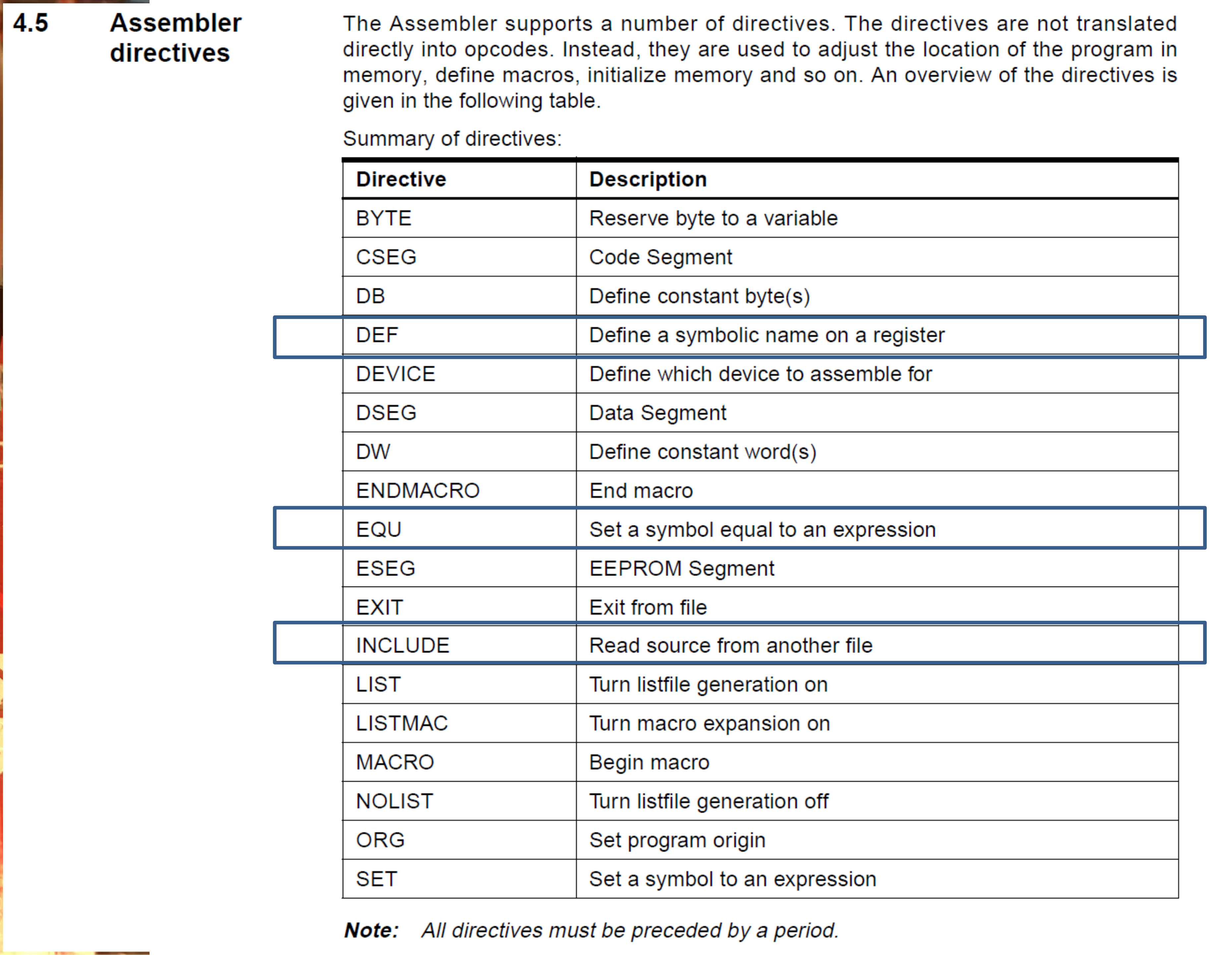
The Assembler translates assembly source code into object code. The generated object code can be used as input to a simulator or an emulator such as the Atmel *AVR* In-Circuit Emulator. The Assembler also generates a PROMable code and an optional EEPROM file which can be programmed directly into the program memory and EEPROM memory of an *AVR* microcontroller.

The Assembler generates fixed code allocations, consequently no linking is necessary.

The Assembler runs under Microsoft Windows 3.11, Microsoft Windows95 and Microsoft Windows NT. In addition, there is an MS-DOS command line version. The Windows version of the program contains an on-line help function covering most of this document.

The instruction set of the *AVR* family of microcontrollers is only briefly described, refer to the *AVR* Data Book (also available on CD-ROM) in order to get more detailed knowledge of the instruction set for the different microcontrollers.

To get quickly started, the Quick-Start Tutorial is an easy way to get familiar with the Atmel *AVR* Assembler.



4.5 Assembler directives

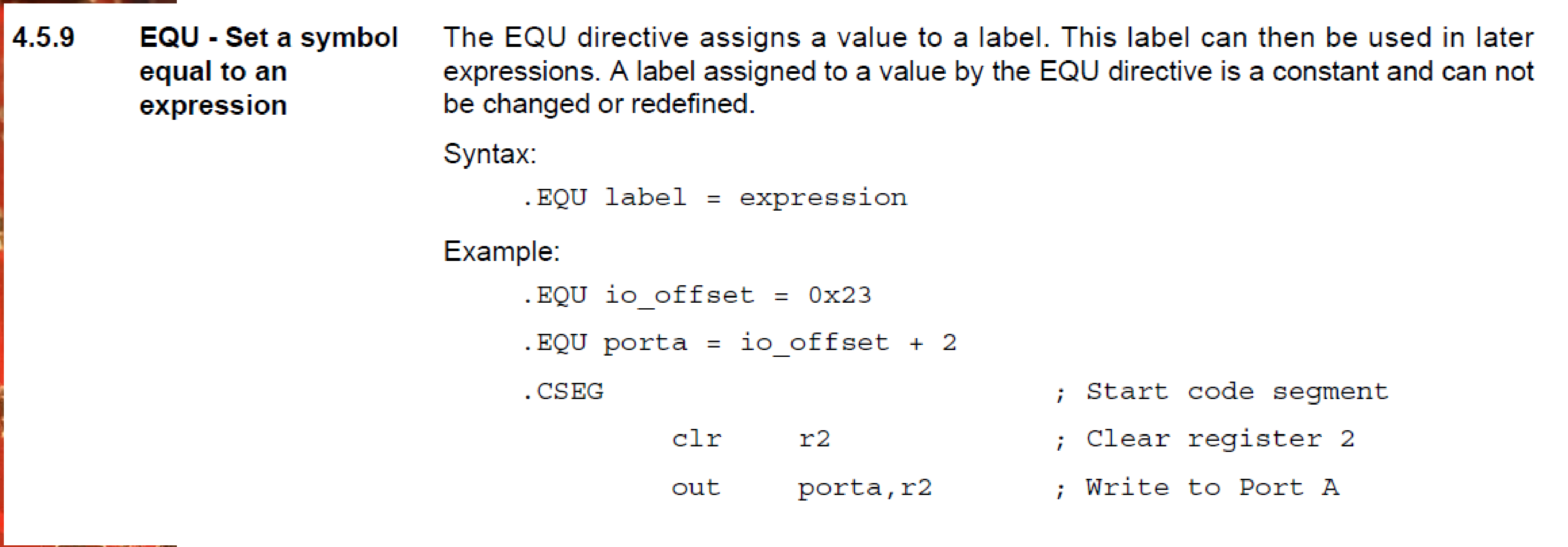
The Assembler supports a number of directives. The directives are not translated directly into opcodes. Instead, they are used to adjust the location of the program in memory, define macros, initialize memory and so on. An overview of the directives is given in the following table.

Summary of directives:

Directive	Description
BYTE	Reserve byte to a variable
CSEG	Code Segment
DB	Define constant byte(s)
DEF	Define a symbolic name on a register
DEVICE	Define which device to assemble for
DSEG	Data Segment
DW	Define constant word(s)
ENDMACRO	End macro
EQU	Set a symbol equal to an expression
ESEG	EEPROM Segment
EXIT	Exit from file
INCLUDE	Read source from another file
LIST	Turn listfile generation on
LISTMAC	Turn macro expansion on
MACRO	Begin macro
NOLIST	Turn listfile generation off
ORG	Set program origin
SET	Set a symbol to an expression

Note: *All directives must be preceded by a period.*

.EQU



4.5.9 EQU - Set a symbol equal to an expression

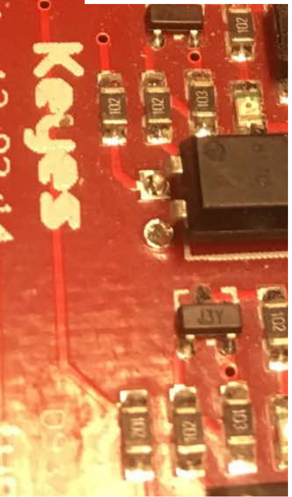
The EQU directive assigns a value to a label. This label can then be used in later expressions. A label assigned to a value by the EQU directive is a constant and can not be changed or redefined.

Syntax:

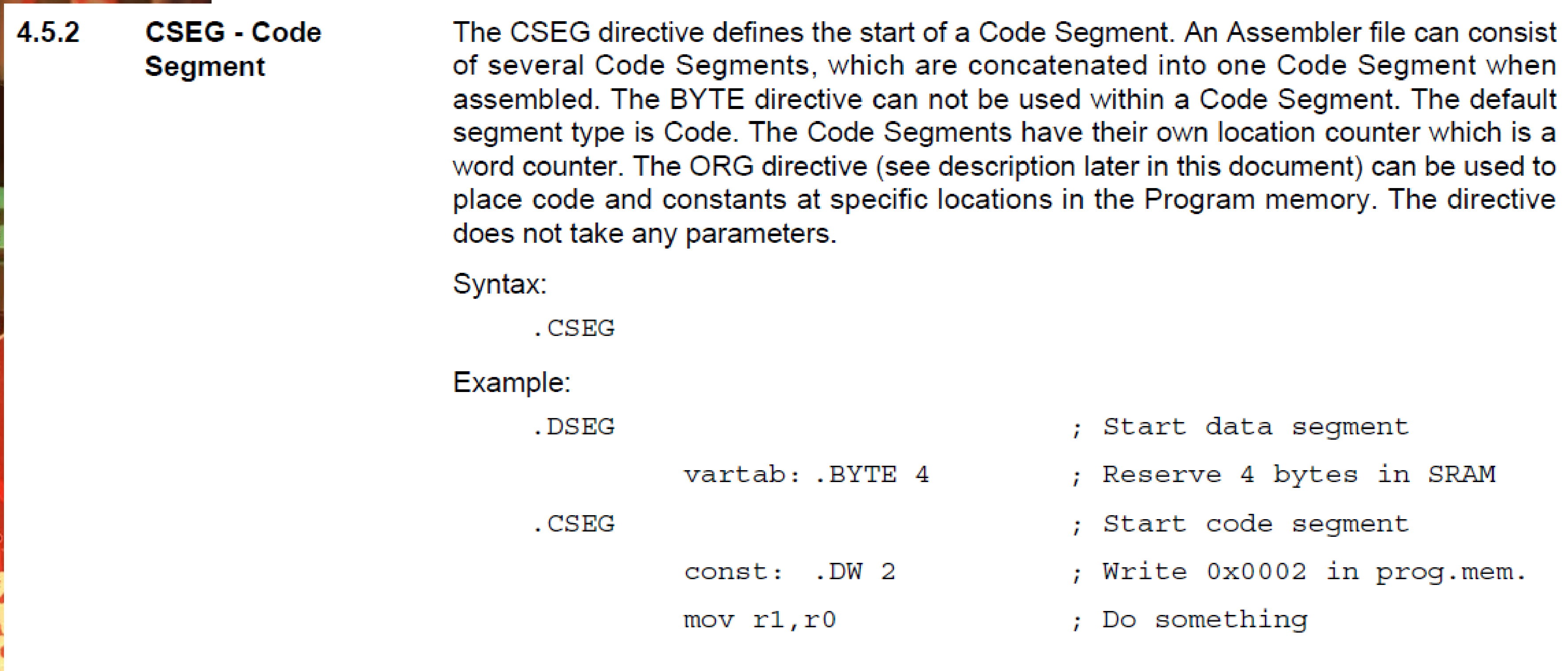
```
.EQU label = expression
```

Example:

```
.EQU io_offset = 0x23
.EQU porta = io_offset + 2
.CSEG                                   ; Start code segment
        clr     r2                      ; Clear register 2
        out     porta,r2                ; Write to Port A
```



.CSEG



4.5.2 CSEG - Code Segment

The CSEG directive defines the start of a Code Segment. An Assembler file can consist of several Code Segments, which are concatenated into one Code Segment when assembled. The BYTE directive can not be used within a Code Segment. The default segment type is Code. The Code Segments have their own location counter which is a word counter. The ORG directive (see description later in this document) can be used to place code and constants at specific locations in the Program memory. The directive does not take any parameters.

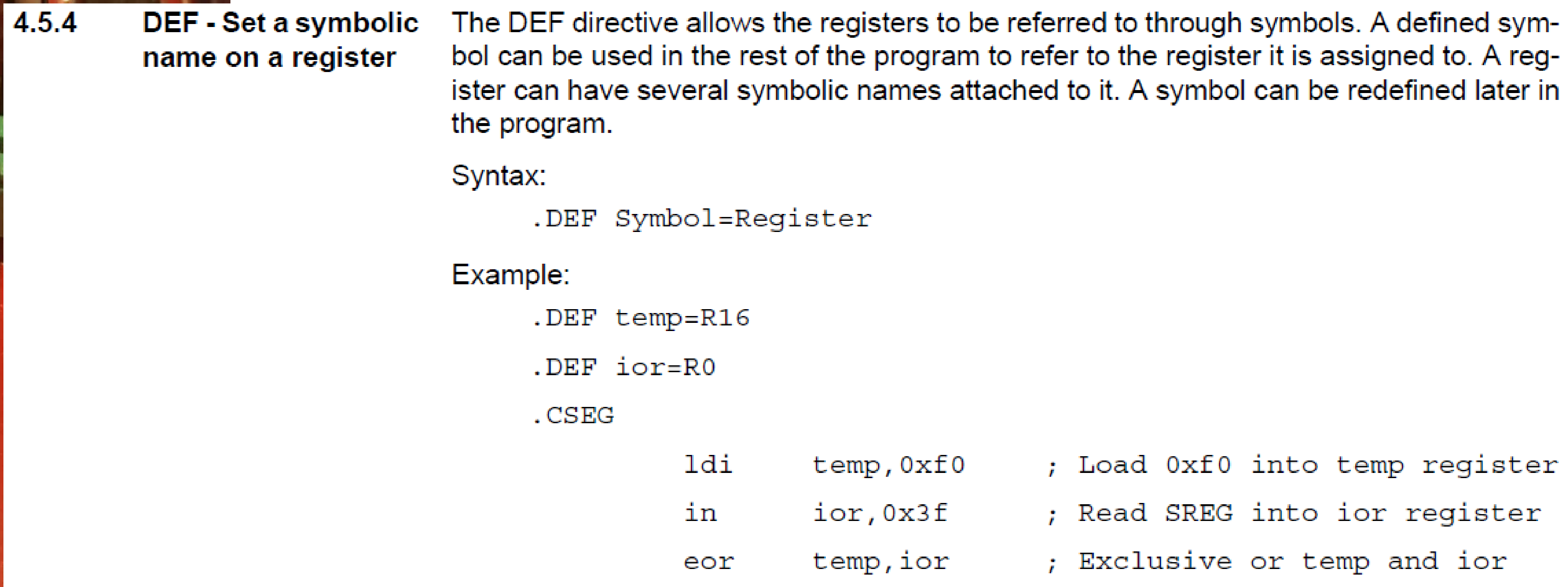
Syntax:

```
.CSEG
```

Example:

```
.DSEG                              ; Start data segment
          vartab: .BYTE 4          ; Reserve 4 bytes in SRAM
.CSEG                              ; Start code segment
          const:  .DW 2            ; Write 0x0002 in prog.mem.
          mov r1,r0                ; Do something
```

.DEF



4.5.4 DEF - Set a symbolic name on a register

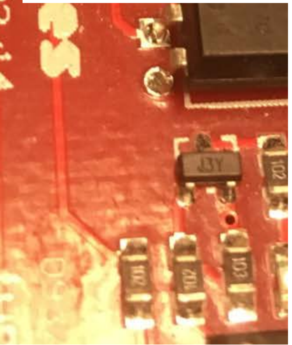
The DEF directive allows the registers to be referred to through symbols. A defined symbol can be used in the rest of the program to refer to the register it is assigned to. A register can have several symbolic names attached to it. A symbol can be redefined later in the program.

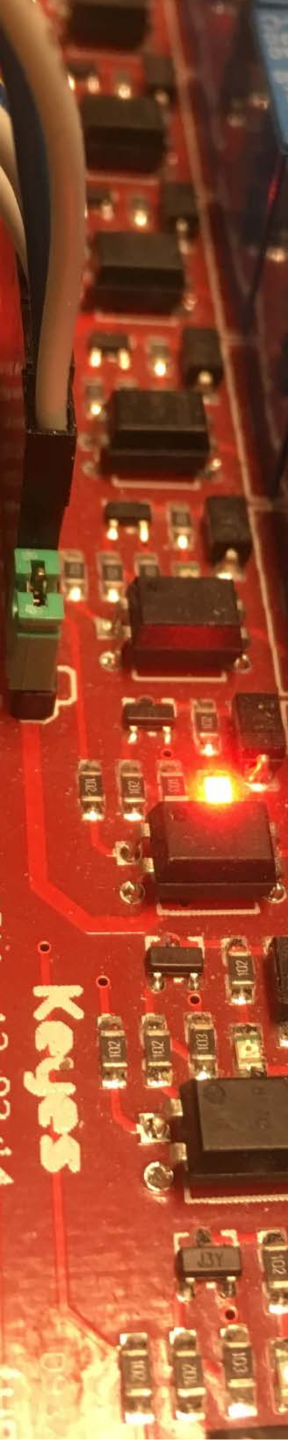
Syntax:

```
.DEF Symbol=Register
```

Example:

```
.DEF temp=R16
.DEF ior=R0
.CSEG
         ldi    temp,0xf0     ; Load 0xf0 into temp register
         in     ior,0x3f      ; Read SREG into ior register
         eor    temp,ior      ; Exclusive or temp and ior
```

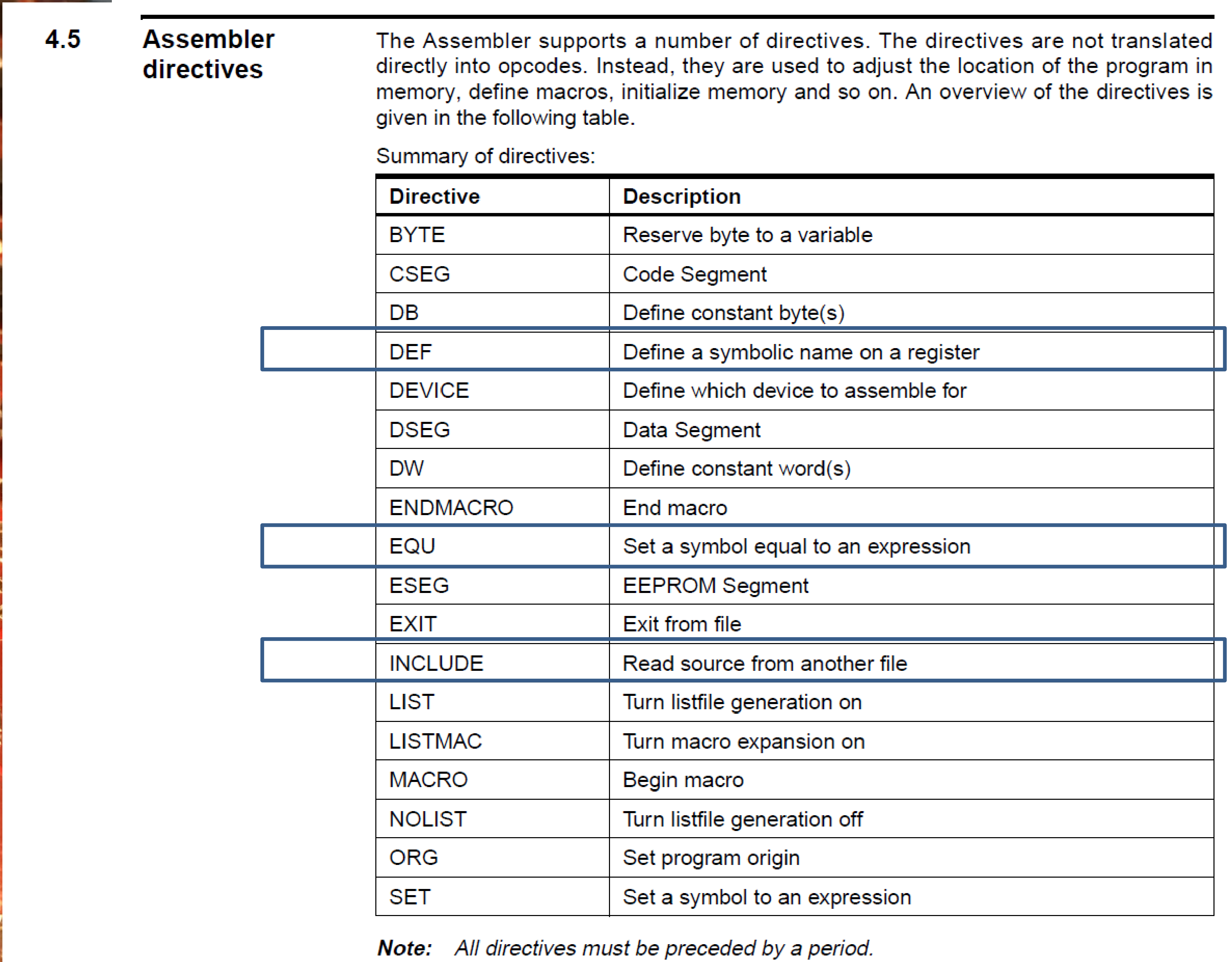




Assembler commands:

- Example:

```
.def Temp = r16
.equ PINA = 0x00
.equ DDRA = 0x01
.cseg
```



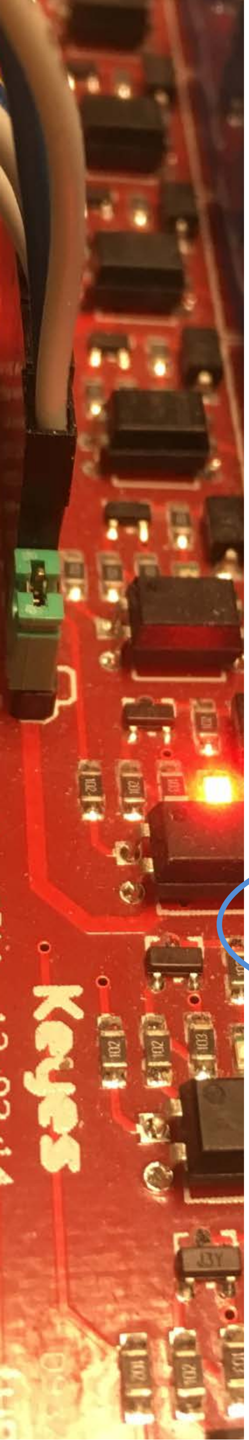
4.5 Assembler directives

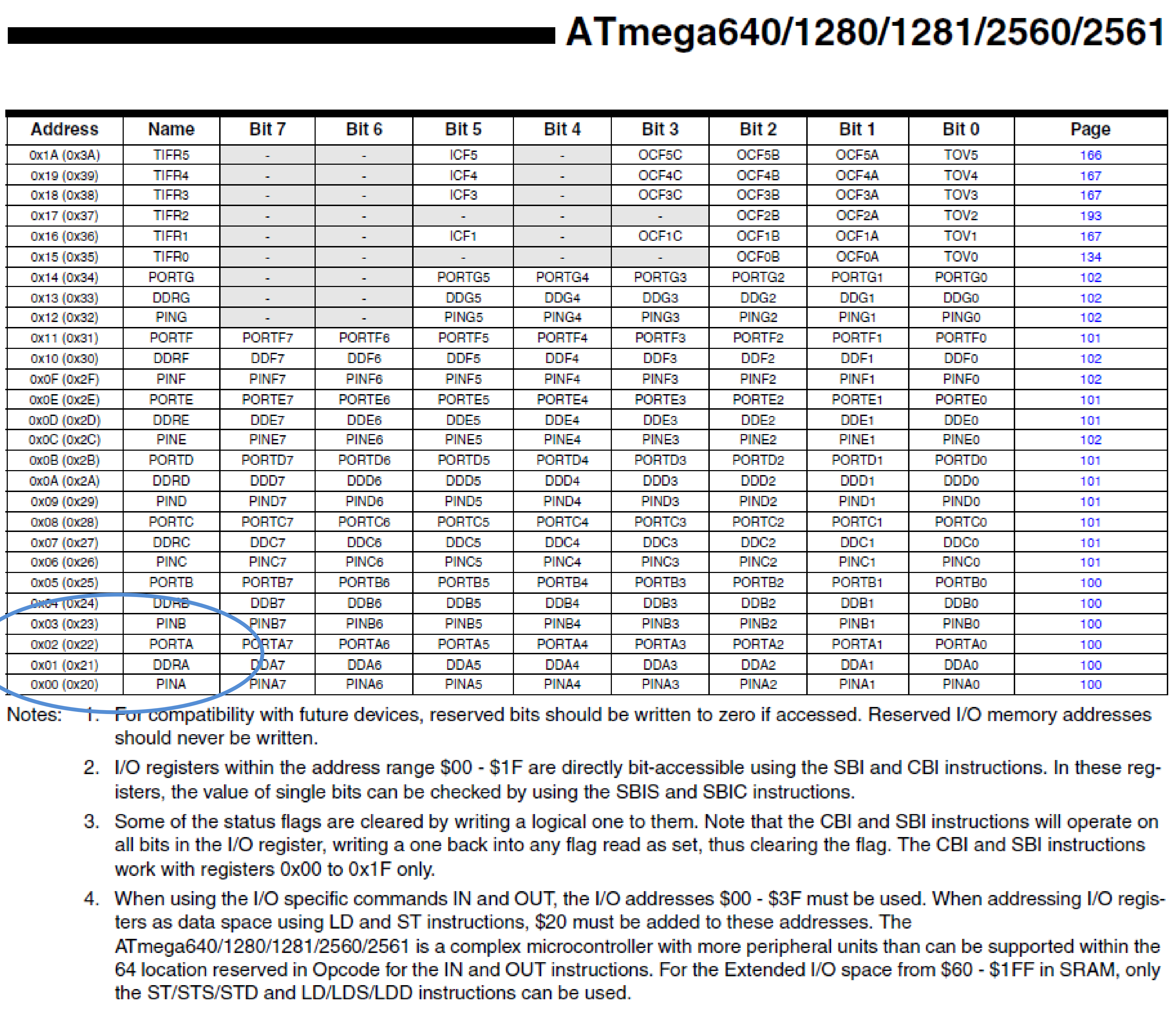
The Assembler supports a number of directives. The directives are not translated directly into opcodes. Instead, they are used to adjust the location of the program in memory, define macros, initialize memory and so on. An overview of the directives is given in the following table.

Summary of directives:

Directive	Description
BYTE	Reserve byte to a variable
CSEG	Code Segment
DB	Define constant byte(s)
DEF	Define a symbolic name on a register
DEVICE	Define which device to assemble for
DSEG	Data Segment
DW	Define constant word(s)
ENDMACRO	End macro
EQU	Set a symbol equal to an expression
ESEG	EEPROM Segment
EXIT	Exit from file
INCLUDE	Read source from another file
LIST	Turn listfile generation on
LISTMAC	Turn macro expansion on
MACRO	Begin macro
NOLIST	Turn listfile generation off
ORG	Set program origin
SET	Set a symbol to an expression

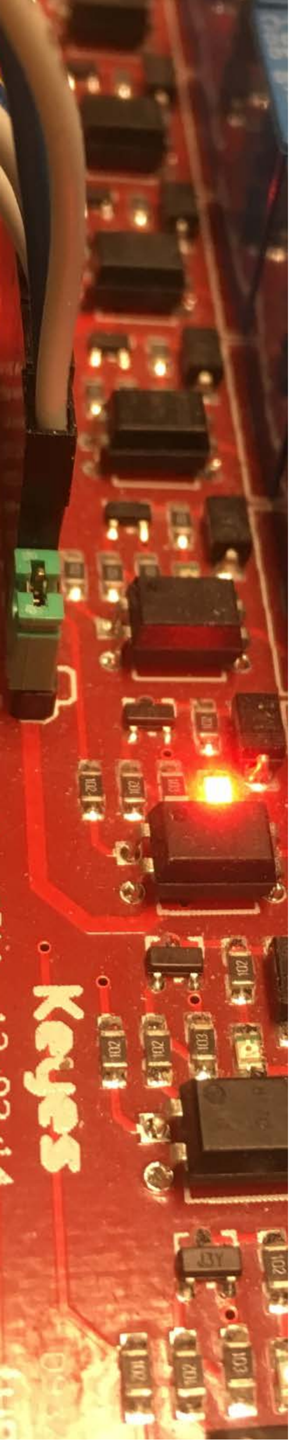
Note: *All directives must be preceded by a period.*

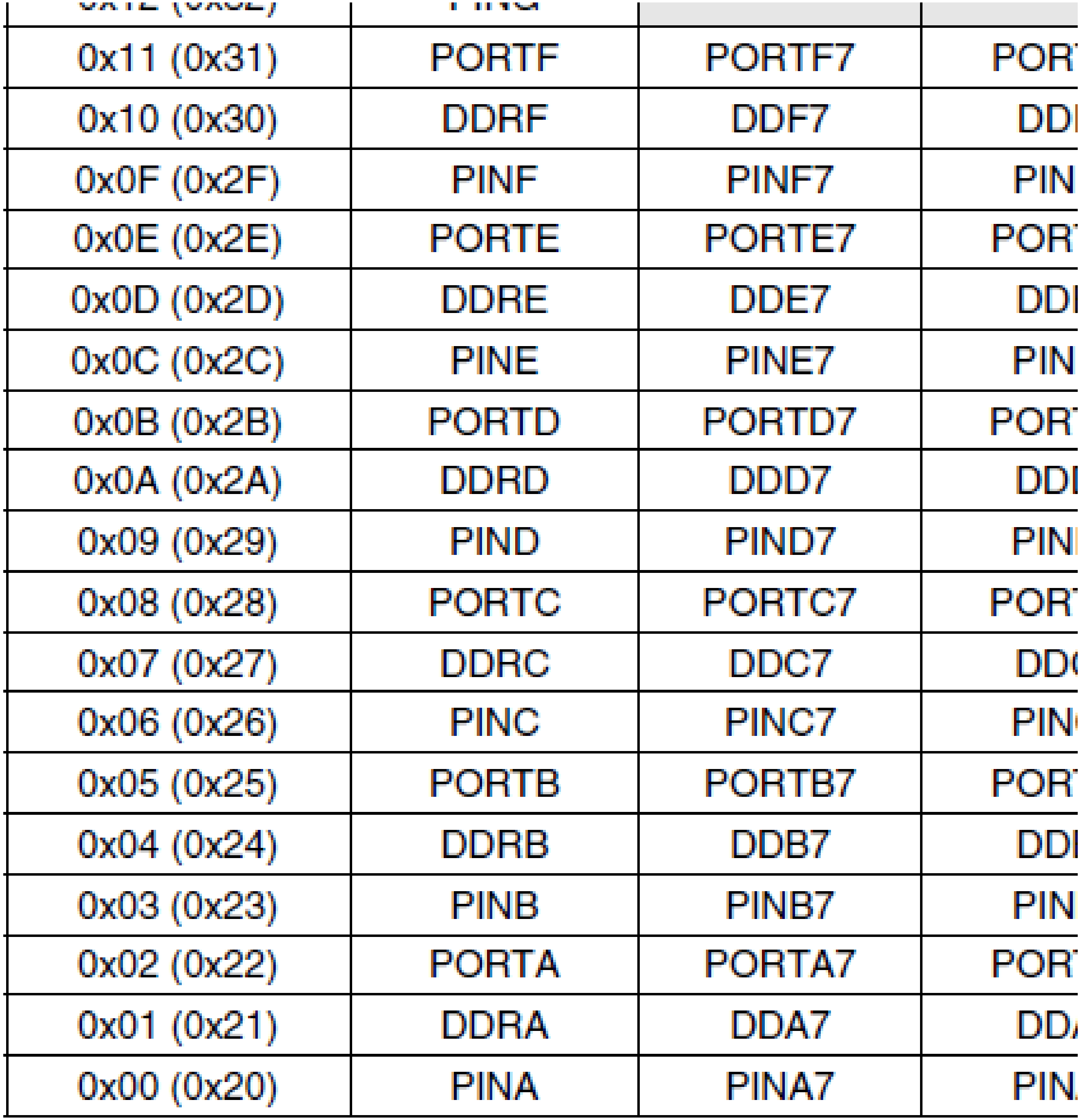




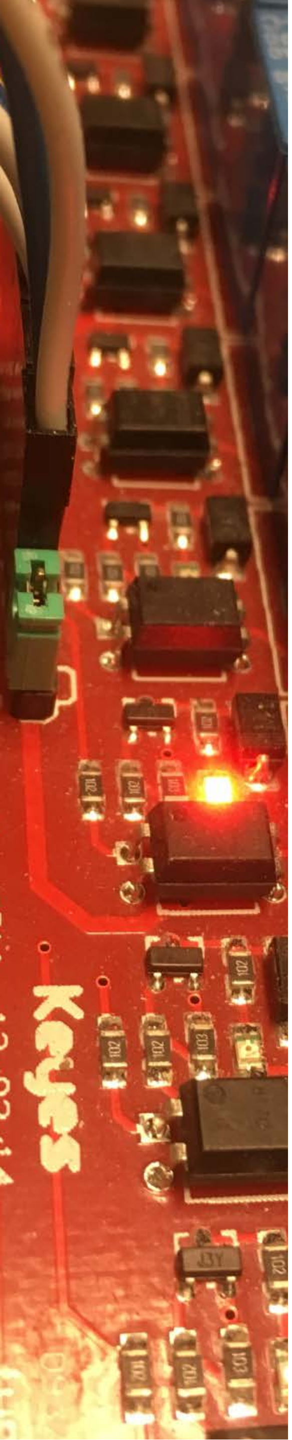
Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x1A (0x3A)	TIFR5	-	-	ICF5	-	OCF5C	OCF5B	OCF5A	TOV5	166
0x19 (0x39)	TIFR4	-	-	ICF4	-	OCF4C	OCF4B	OCF4A	TOV4	167
0x18 (0x38)	TIFR3	-	-	ICF3	-	OCF3C	OCF3B	OCF3A	TOV3	167
0x17 (0x37)	TIFR2	-	-	-	-	-	OCF2B	OCF2A	TOV2	193
0x16 (0x36)	TIFR1	-	-	ICF1	-	OCF1C	OCF1B	OCF1A	TOV1	167
0x15 (0x35)	TIFR0	-	-	-	-	-	OCF0B	OCF0A	TOV0	134
0x14 (0x34)	PORTG	-	-	PORTG5	PORTG4	PORTG3	PORTG2	PORTG1	PORTG0	102
0x13 (0x33)	DDRG	-	-	DDG5	DDG4	DDG3	DDG2	DDG1	DDG0	102
0x12 (0x32)	PING	-	-	PING5	PING4	PING3	PING2	PING1	PING0	102
0x11 (0x31)	PORTF	PORTF7	PORTF6	PORTF5	PORTF4	PORTF3	PORTF2	PORTF1	PORTF0	101
0x10 (0x30)	DDRF	DDF7	DDF6	DDF5	DDF4	DDF3	DDF2	DDF1	DDF0	102
0x0F (0x2F)	PINF	PINF7	PINF6	PINF5	PINF4	PINF3	PINF2	PINF1	PINF0	102
0x0E (0x2E)	PORTE	PORTE7	PORTE6	PORTE5	PORTE4	PORTE3	PORTE2	PORTE1	PORTE0	101
0x0D (0x2D)	DDRE	DDE7	DDE6	DDE5	DDE4	DDE3	DDE2	DDE1	DDE0	101
0x0C (0x2C)	PINE	PINE7	PINE6	PINE5	PINE4	PINE3	PINE2	PINE1	PINE0	102
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	101
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	101
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	101
0x08 (0x28)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	101
0x07 (0x27)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	101
0x06 (0x26)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	101
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	100
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	100
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	100
0x02 (0x22)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	100
0x01 (0x21)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	100
0x00 (0x20)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	100

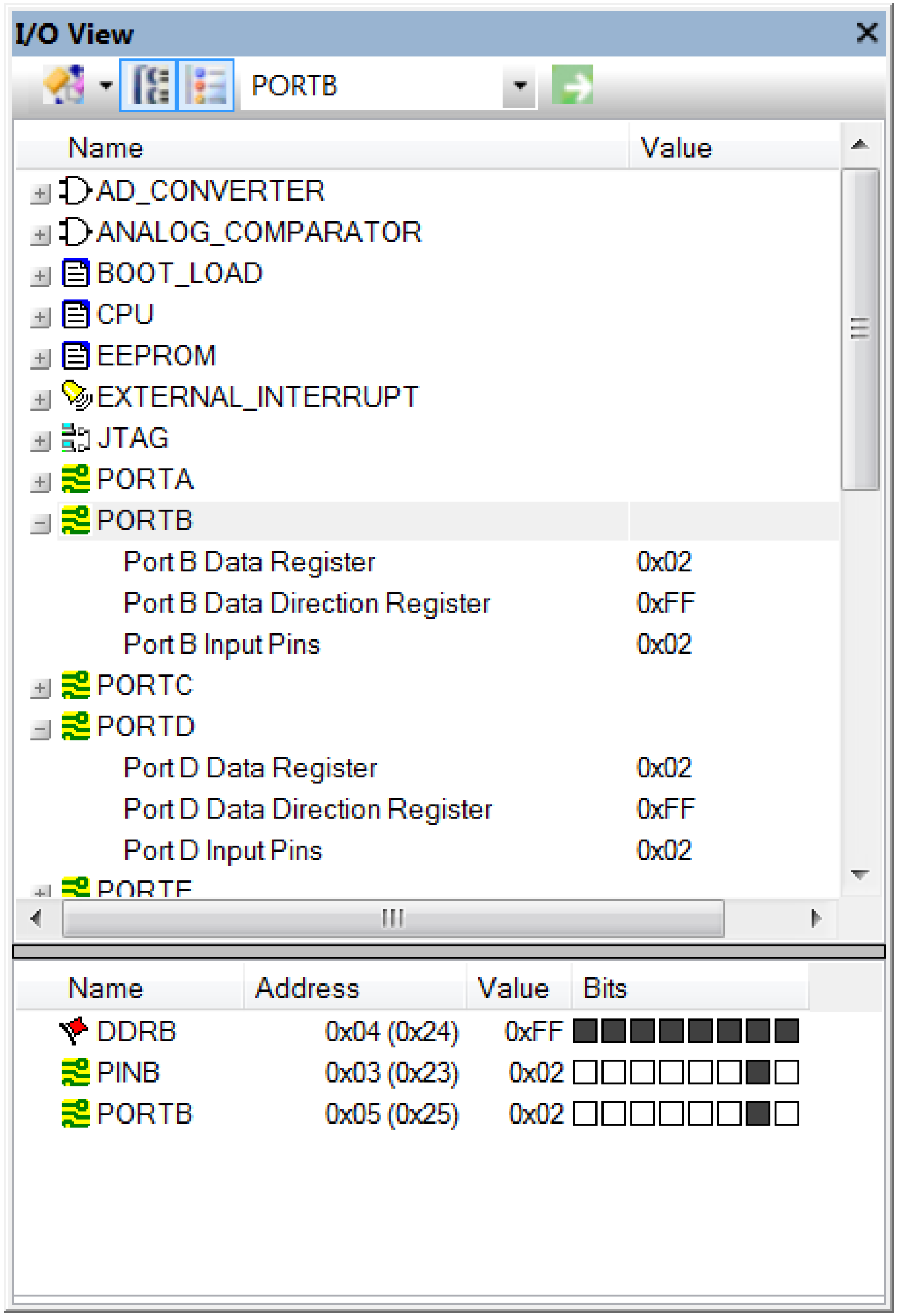
Notes:
1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
2. I/O registers within the address range $00 - $1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions.
3. Some of the status flags are cleared by writing a logical one to them. Note that the CBI and SBI instructions will operate on all bits in the I/O register, writing a one back into any flag read as set, thus clearing the flag. The CBI and SBI instructions work with registers 0x00 to 0x1F only.
4. When using the I/O specific commands IN and OUT, the I/O addresses $00 - $3F must be used. When addressing I/O registers as data space using LD and ST instructions, $20 must be added to these addresses. The ATmega640/1280/1281/2560/2561 is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in Opcode for the IN and OUT instructions. For the Extended I/O space from $60 - $1FF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.





0x12 (0x32)	PING		
0x11 (0x31)	PORTF	PORTF7	POR
0x10 (0x30)	DDRF	DDF7	DD
0x0F (0x2F)	PINF	PINF7	PIN
0x0E (0x2E)	PORTE	PORTE7	POR
0x0D (0x2D)	DDRE	DDE7	DD
0x0C (0x2C)	PINE	PINE7	PIN
0x0B (0x2B)	PORTD	PORTD7	POR
0x0A (0x2A)	DDRD	DDD7	DD
0x09 (0x29)	PIND	PIND7	PIN
0x08 (0x28)	PORTC	PORTC7	POR
0x07 (0x27)	DDRC	DDC7	DD
0x06 (0x26)	PINC	PINC7	PIN
0x05 (0x25)	PORTB	PORTB7	POR
0x04 (0x24)	DDRB	DDB7	DD
0x03 (0x23)	PINB	PINB7	PIN
0x02 (0x22)	PORTA	PORTA7	POR
0x01 (0x21)	DDRA	DDA7	DD
0x00 (0x20)	PINA	PINA7	PIN



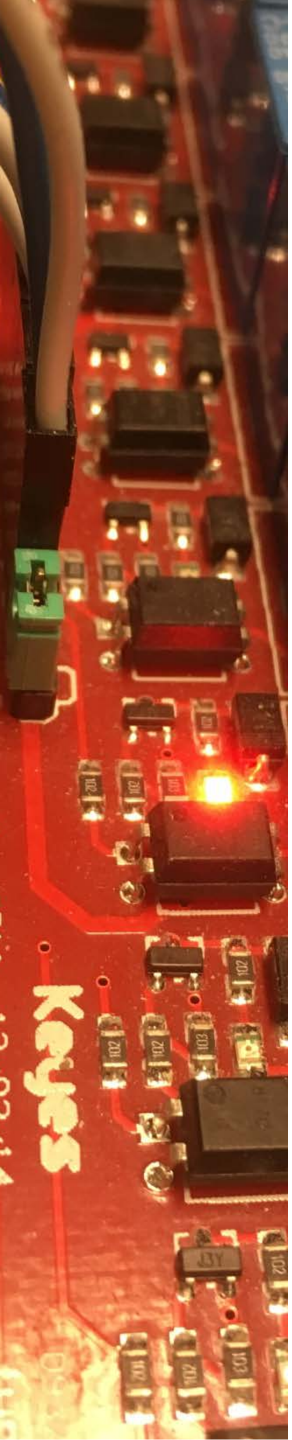


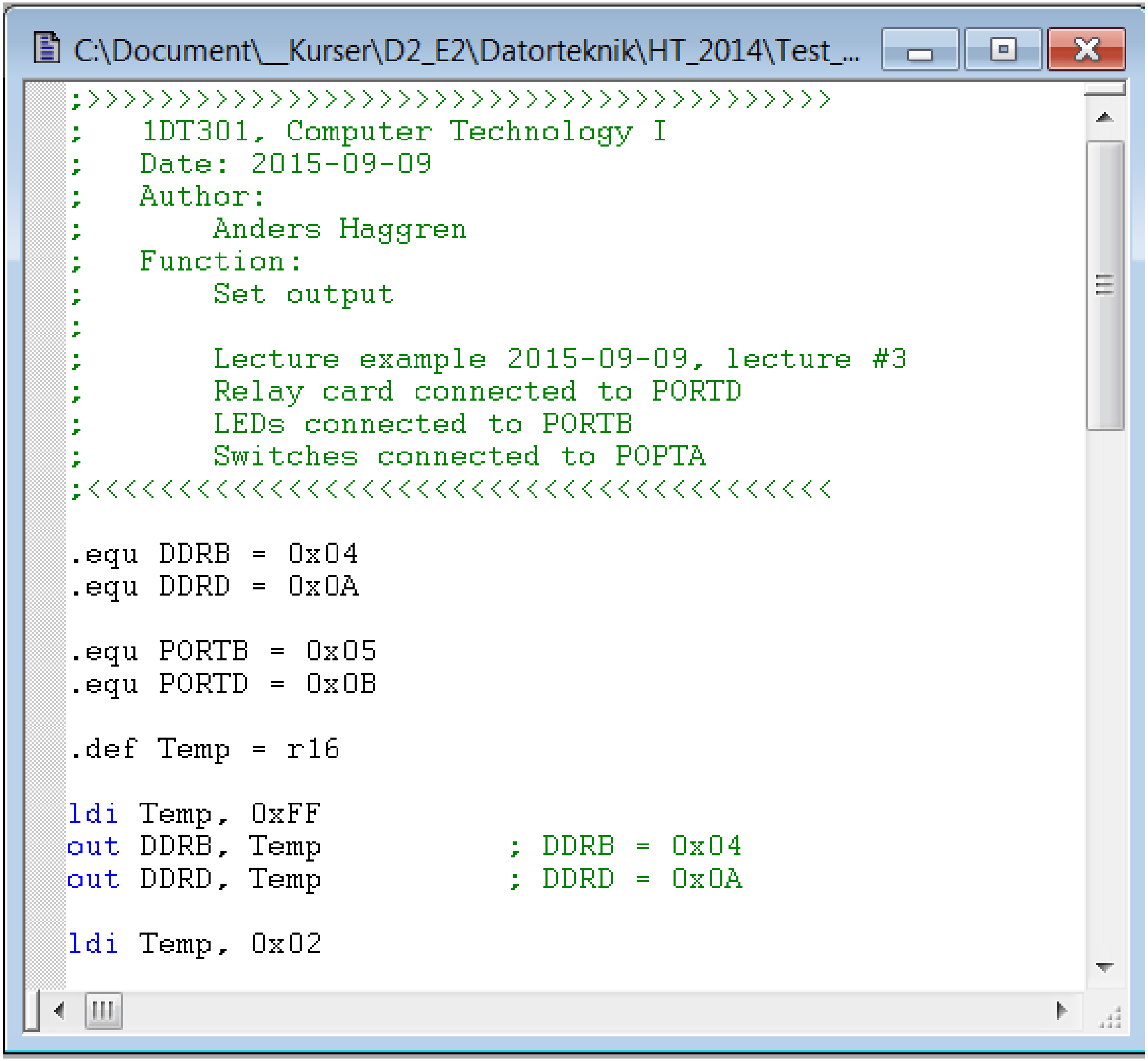
I/O View

PORTB

Name	Value
AD_CONVERTER	
ANALOG_COMPARATOR	
BOOT_LOAD	
CPU	
EEPROM	
EXTERNAL_INTERRUPT	
JTAG	
PORTA	
PORTB	
Port B Data Register	0x02
Port B Data Direction Register	0xFF
Port B Input Pins	0x02
PORTC	
PORTD	
Port D Data Register	0x02
Port D Data Direction Register	0xFF
Port D Input Pins	0x02
PORTE	

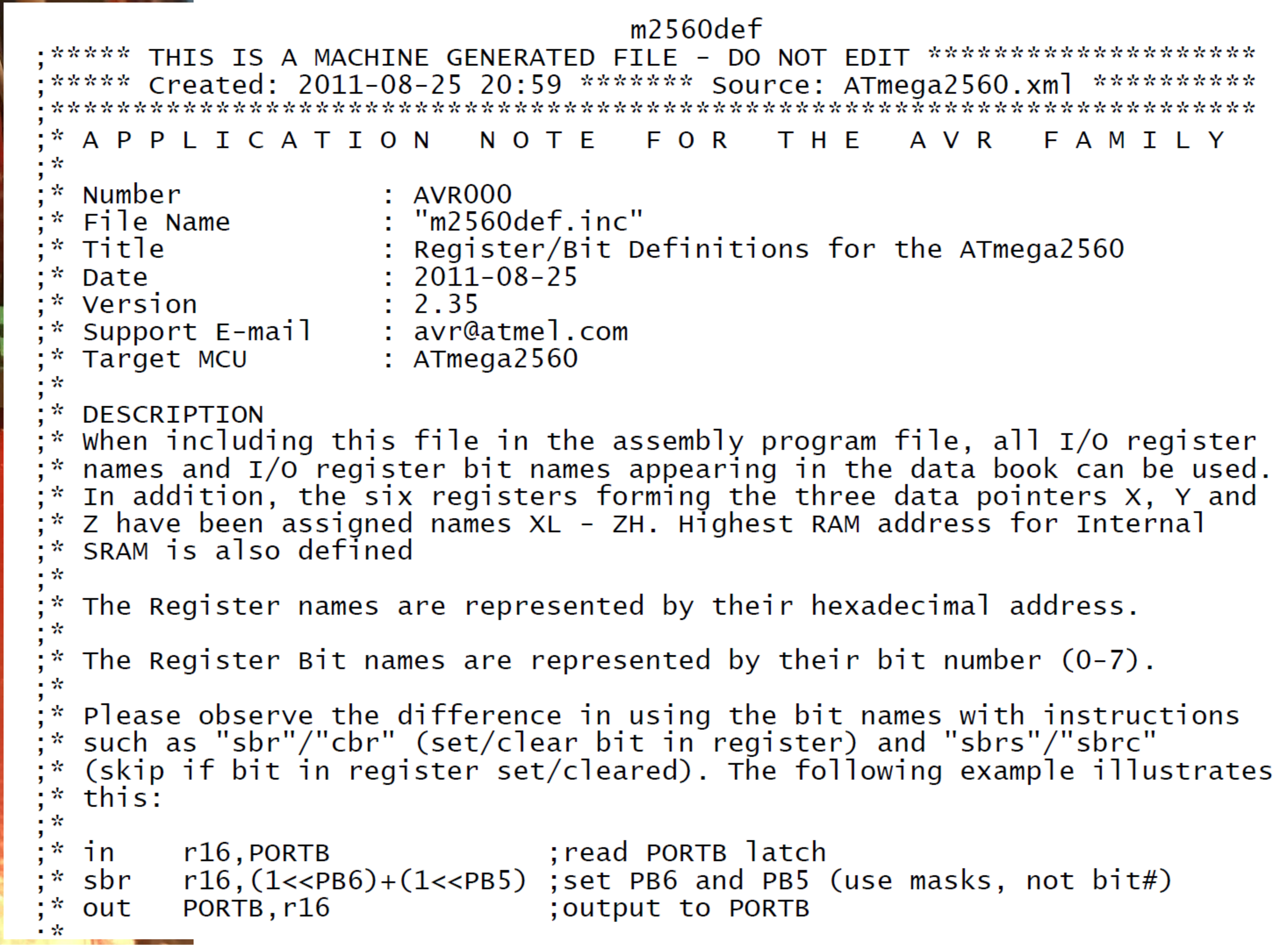
Name	Address	Value	Bits
DDRB	0x04 (0x24)	0xFF	■■■■■■■■
PINB	0x03 (0x23)	0x02	□□□□□□■□
PORTB	0x05 (0x25)	0x02	□□□□□□■□



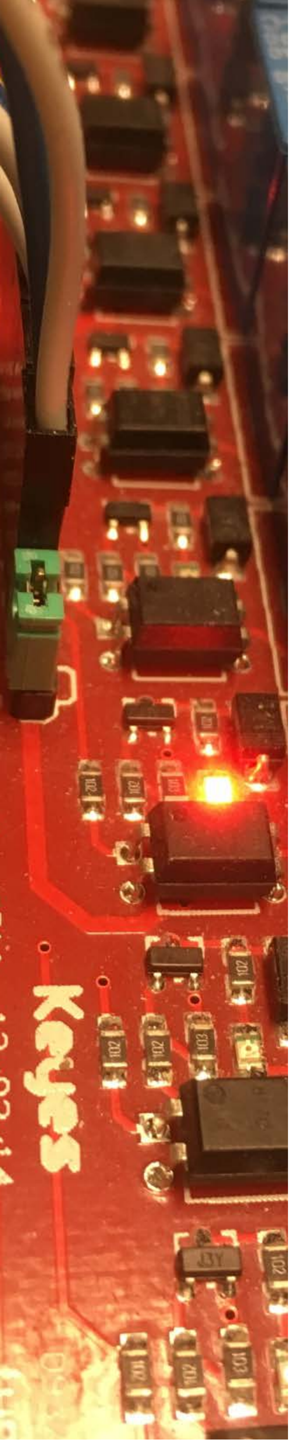


```
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;   1DT301, Computer Technology I
;   Date: 2015-09-09
;   Author:
;       Anders Haggren
;   Function:
;       Set output
;
;       Lecture example 2015-09-09, lecture #3
;       Relay card connected to PORTD
;       LEDs connected to PORTB
;       Switches connected to POPTA
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<


.equ DDRB = 0x04
.equ DDRD = 0x0A

.equ PORTB = 0x05
.equ PORTD = 0x0B

.def Temp = r16

ldi Temp, 0xFF
out DDRB, Temp          ; DDRB = 0x04
out DDRD, Temp          ; DDRD = 0x0A

ldi Temp, 0x02
```

m2560def

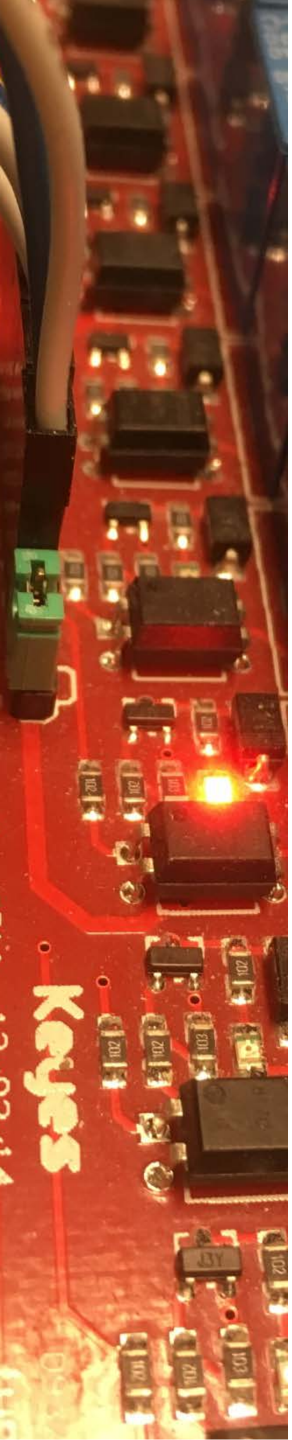


```
                                m2560def
;***** THIS IS A MACHINE GENERATED FILE - DO NOT EDIT ********************
;***** Created: 2011-08-25 20:59 ******* Source: ATmega2560.xml **********
;*************************************************************************
;* A P P L I C A T I O N   N O T E   F O R   T H E   A V R   F A M I L Y
;*
;* Number             : AVR000
;* File Name          : "m2560def.inc"
;* Title              : Register/Bit Definitions for the ATmega2560
;* Date               : 2011-08-25
;* Version            : 2.35
;* Support E-mail     : avr@atmel.com
;* Target MCU         : ATmega2560
;*
;* DESCRIPTION
;* When including this file in the assembly program file, all I/O register
;* names and I/O register bit names appearing in the data book can be used.
;* In addition, the six registers forming the three data pointers X, Y and
;* Z have been assigned names XL - ZH. Highest RAM address for Internal
;* SRAM is also defined
;*
;* The Register names are represented by their hexadecimal address.
;*
;* The Register Bit names are represented by their bit number (0-7).
;*
;* Please observe the difference in using the bit names with instructions
;* such as "sbr"/"cbr" (set/clear bit in register) and "sbrs"/"sbrc"
;* (skip if bit in register set/cleared). The following example illustrates
;* this:
;*
;* in    r16,PORTB             ;read PORTB latch
;* sbr   r16,(1<<PB6)+(1<<PB5) ;set PB6 and PB5 (use masks, not bit#)
;* out   PORTB,r16             ;output to PORTB
;*
```

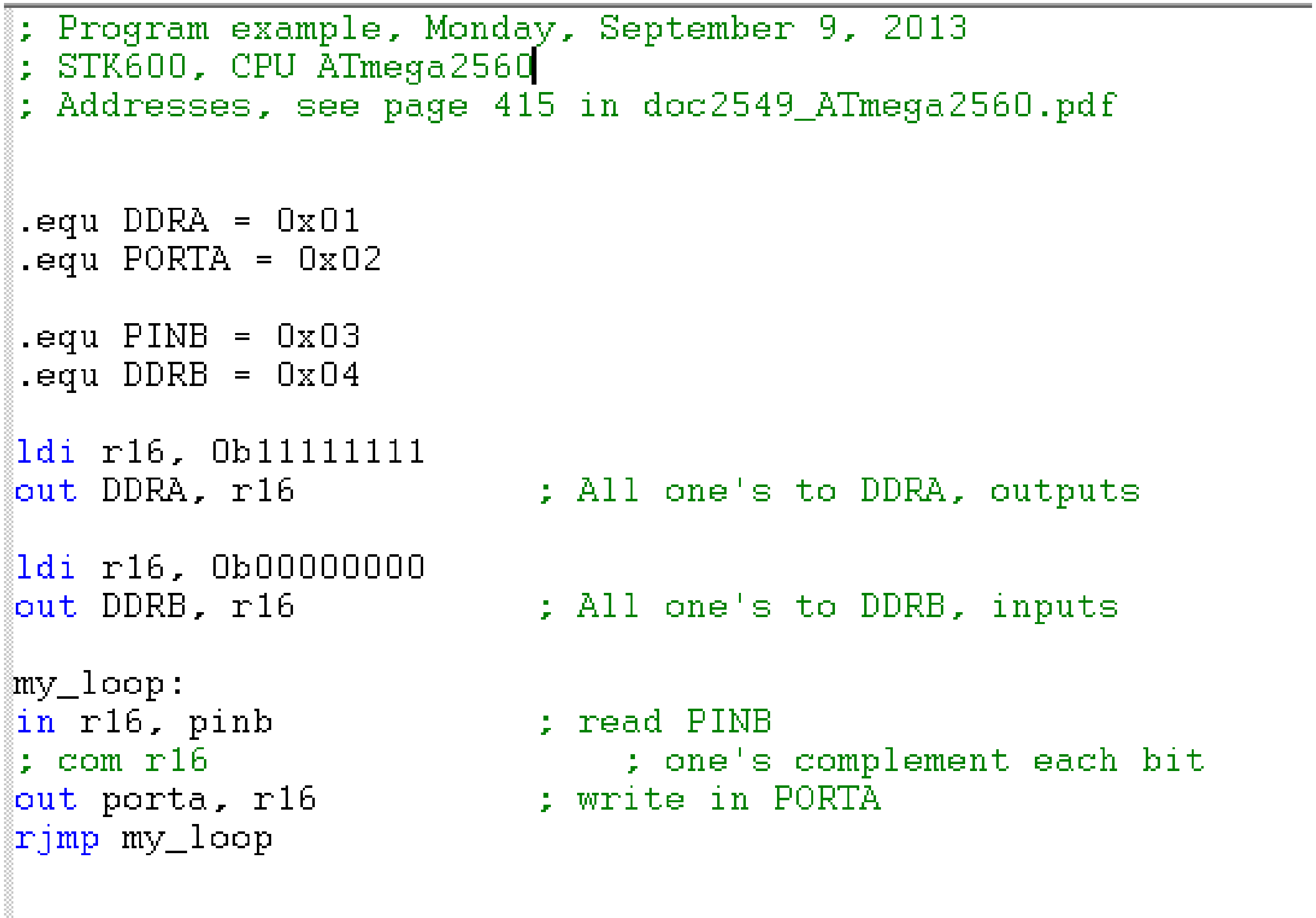


m2560def

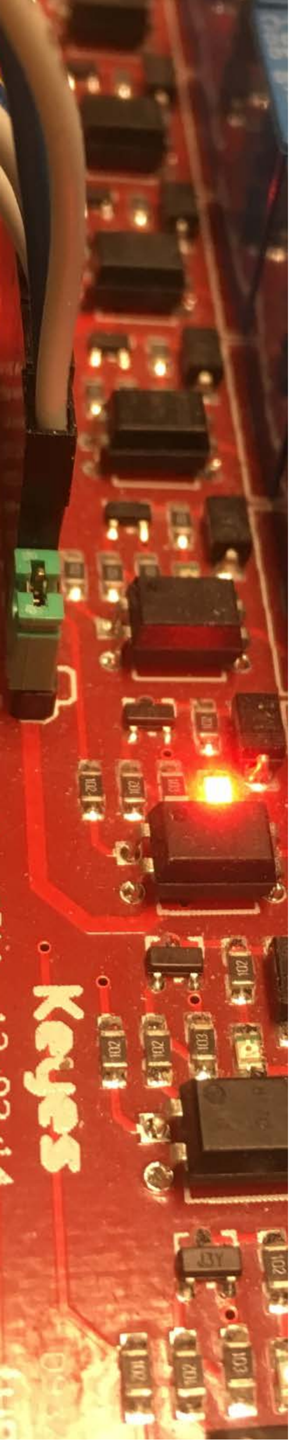
```
.equ    PORTF    =  0x11
.equ    DDRF     =  0x10
.equ    PINF     =  0x0f
.equ    PORTE    =  0x0e
.equ    DDRE     =  0x0d
.equ    PINE     =  0x0c
.equ    PORTD    =  0x0b
.equ    DDRD     =  0x0a
.equ    PIND     =  0x09
.equ    PORTC    =  0x08
.equ    DDRC     =  0x07
.equ    PINC     =  0x06
.equ    PORTB    =  0x05
.equ    DDRB     =  0x04
.equ    PINB     =  0x03
.equ    PORTA    =  0x02
.equ    DDRA     =  0x01
.equ    PINA     =  0x00
```



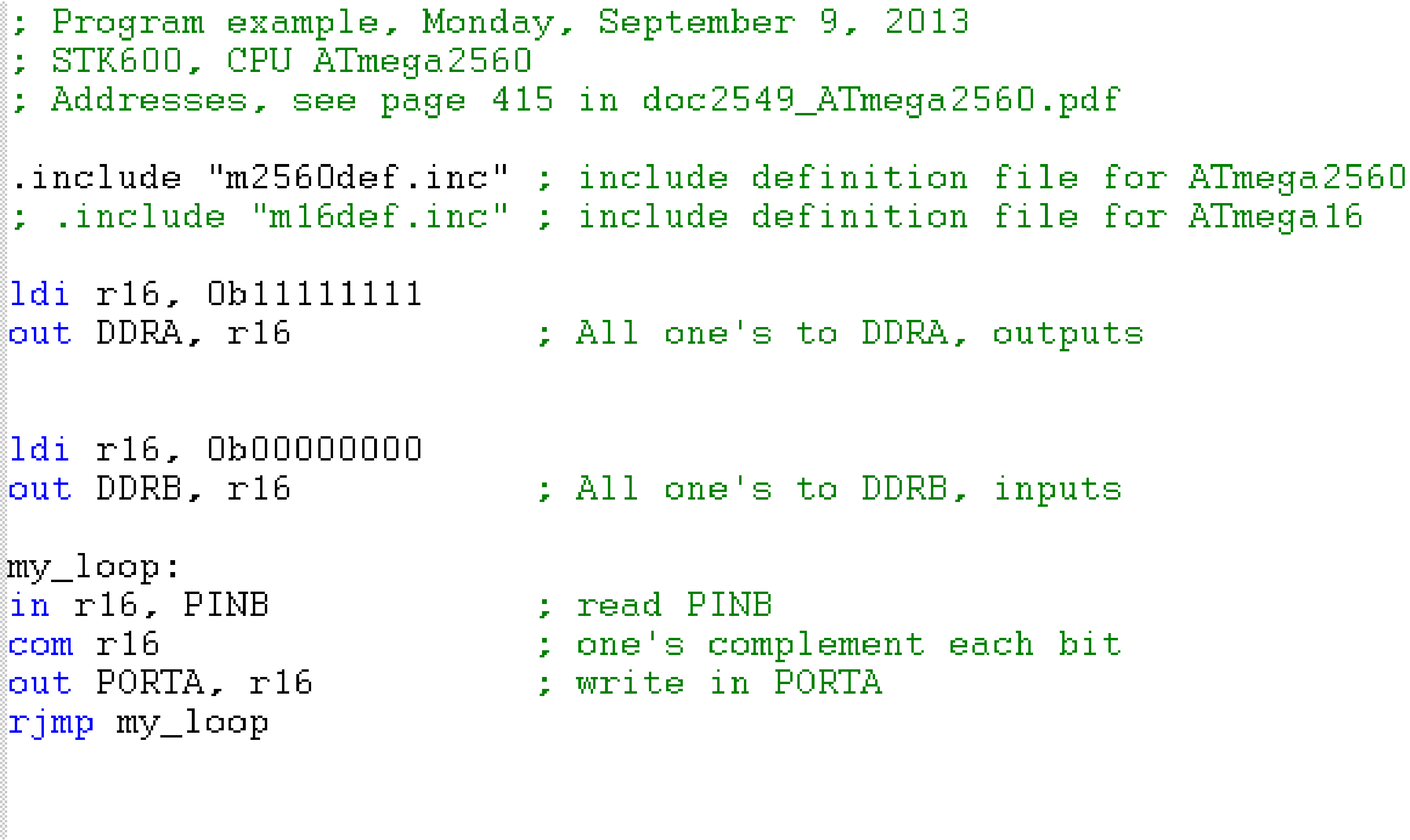
Example with .equ



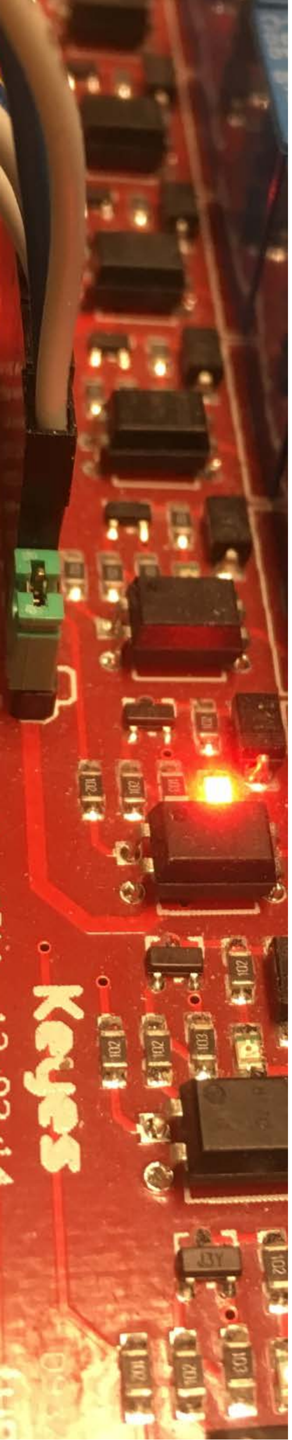
```
; Program example, Monday, September 9, 2013
; STK600, CPU ATmega2560
; Addresses, see page 415 in doc2549_ATmega2560.pdf


.equ DDRA = 0x01
.equ PORTA = 0x02

.equ PINB = 0x03
.equ DDRB = 0x04

ldi r16, 0b11111111
out DDRA, r16               ; All one's to DDRA, outputs

ldi r16, 0b00000000
out DDRB, r16               ; All one's to DDRB, inputs

my_loop:
in r16, pinb                ; read PINB
; com r16                       ; one's complement each bit
out porta, r16              ; write in PORTA
rjmp my_loop
```

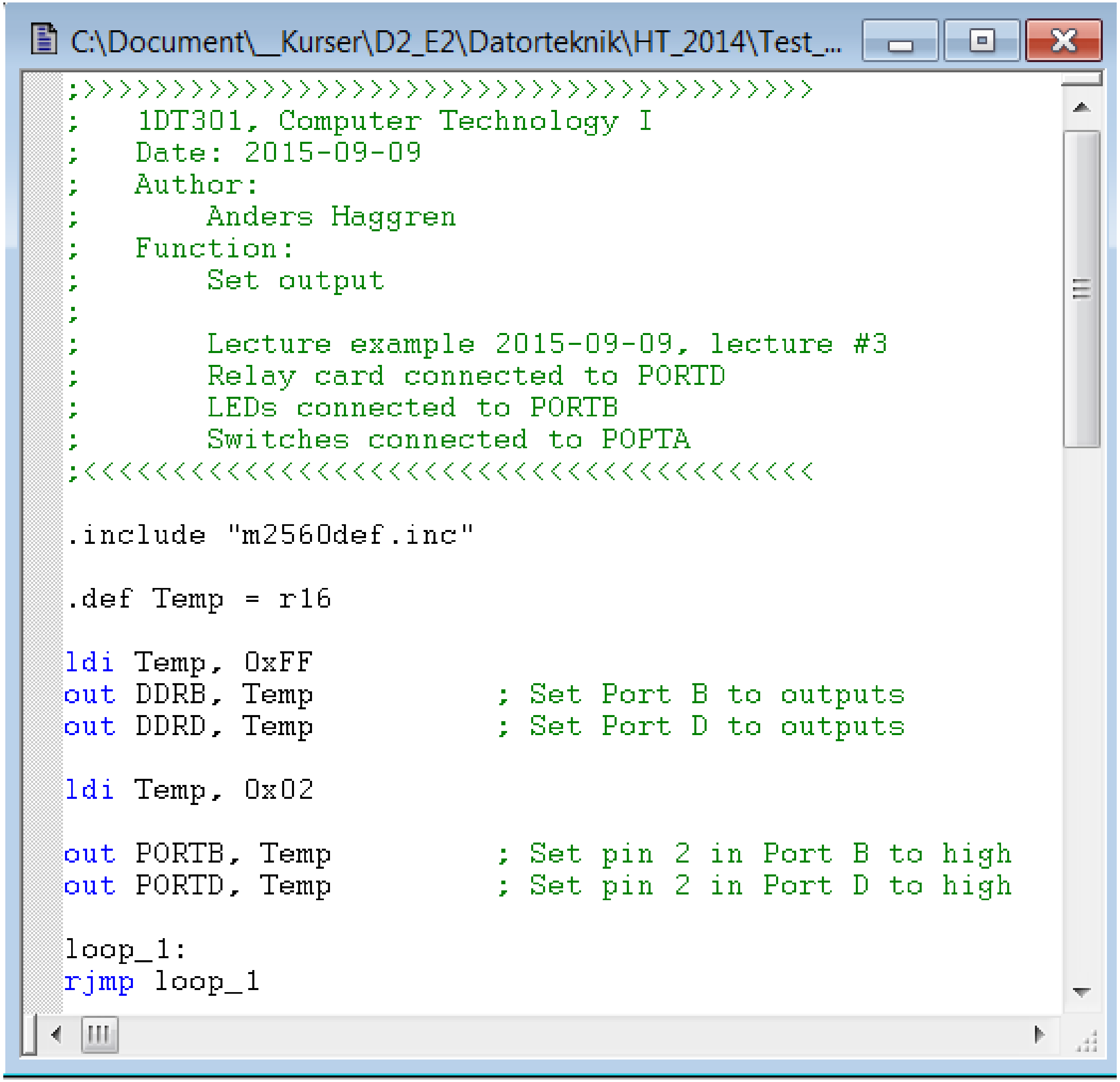


Example with .include ”m2560def.inc”



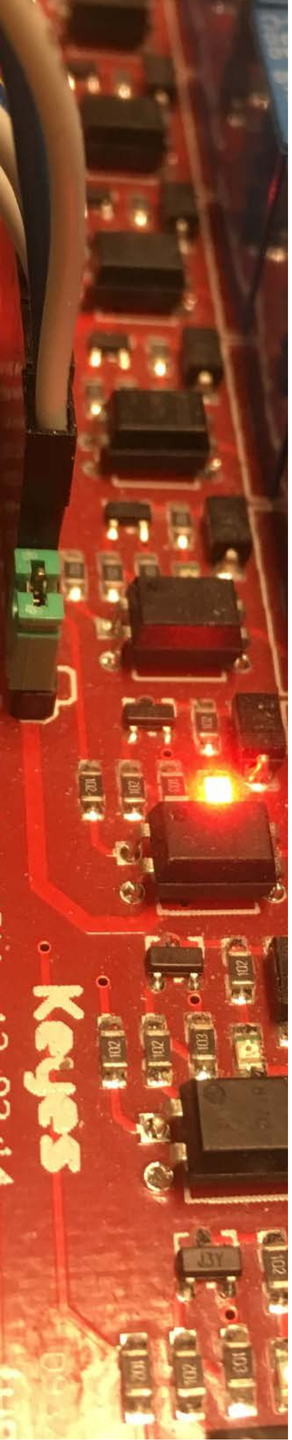
```
; Program example, Monday, September 9, 2013
; STK600, CPU ATmega2560
; Addresses, see page 415 in doc2549_ATmega2560.pdf

.include "m2560def.inc" ; include definition file for ATmega2560
; .include "m16def.inc" ; include definition file for ATmega16

ldi r16, 0b11111111
out DDRA, r16           ; All one's to DDRA, outputs


ldi r16, 0b00000000
out DDRB, r16           ; All one's to DDRB, inputs

my_loop:
in r16, PINB            ; read PINB
com r16                 ; one's complement each bit
out PORTA, r16          ; write in PORTA
rjmp my_loop
```

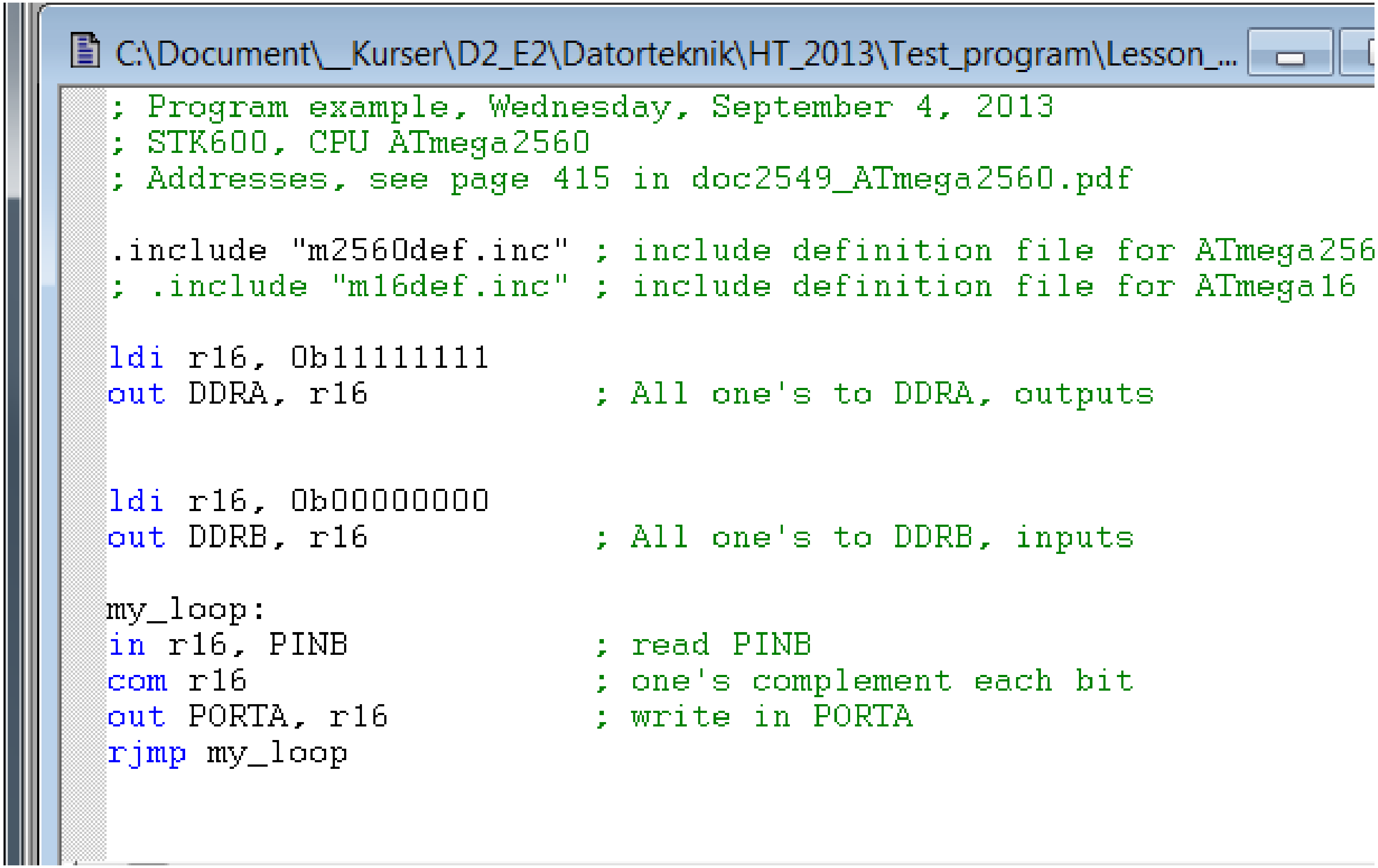




```
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;    1DT301, Computer Technology I
;    Date: 2015-09-09
;    Author:
;        Anders Haggren
;    Function:
;        Set output
;
;        Lecture example 2015-09-09, lecture #3
;        Relay card connected to PORTD
;        LEDs connected to PORTB
;        Switches connected to POPTA
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

.include "m2560def.inc"

.def Temp = r16

ldi Temp, 0xFF
out DDRB, Temp          ; Set Port B to outputs
out DDRD, Temp          ; Set Port D to outputs

ldi Temp, 0x02

out PORTB, Temp         ; Set pin 2 in Port B to high
out PORTD, Temp         ; Set pin 2 in Port D to high

loop_1:
rjmp loop_1
```

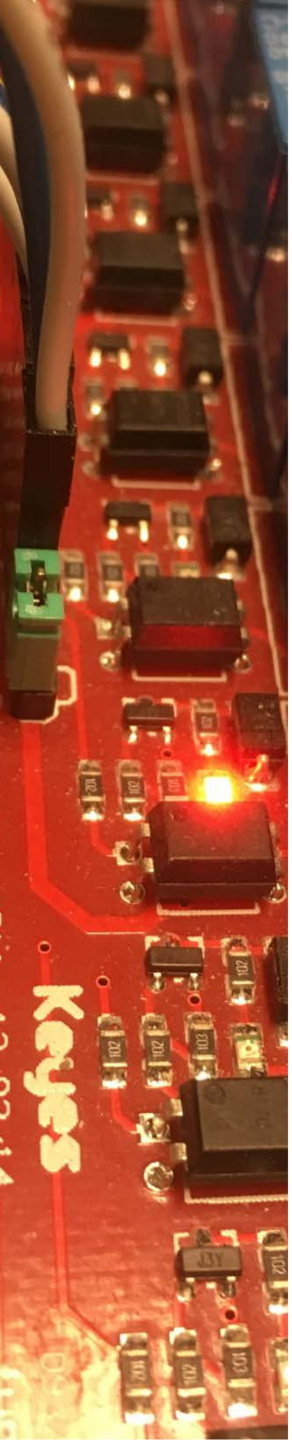
Example with .include ”m2560def.inc”





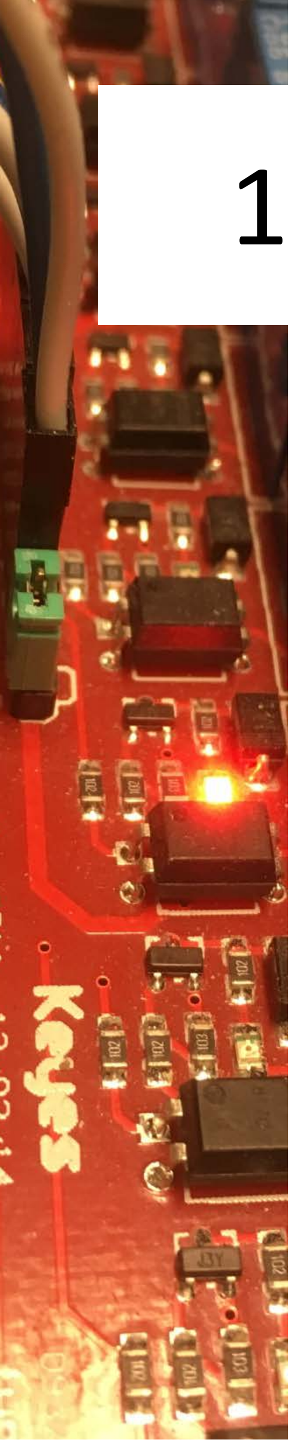
C:\Document_Kurser\D2_E2\Datorteknik\HT_2013\Test_program\Lesson_...

```
; Program example, Wednesday, September 4, 2013
; STK600, CPU ATmega2560
; Addresses, see page 415 in doc2549_ATmega2560.pdf

.include "m2560def.inc" ; include definition file for ATmega256
; .include "m16def.inc" ; include definition file for ATmega16

ldi r16, 0b11111111
out DDRA, r16           ; All one's to DDRA, outputs


ldi r16, 0b00000000
out DDRB, r16           ; All one's to DDRB, inputs

my_loop:
in r16, PINB            ; read PINB
com r16                 ; one's complement each bit
out PORTA, r16          ; write in PORTA
rjmp my_loop
```



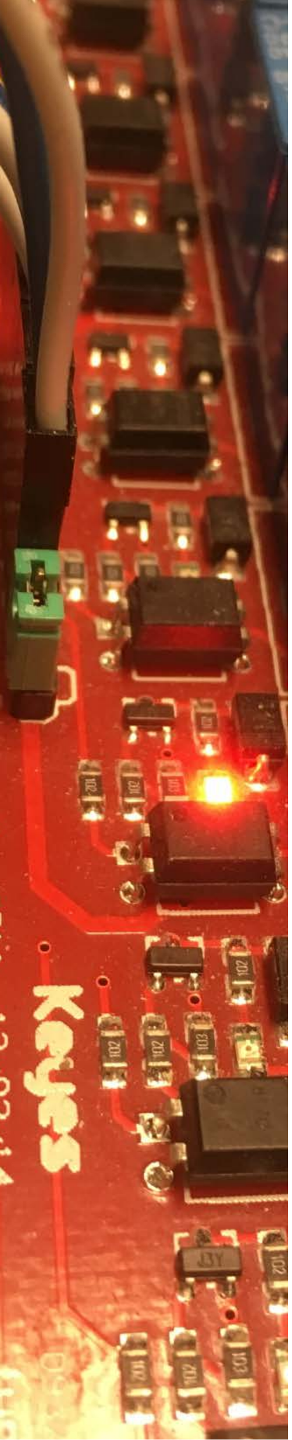
Example from lecture #2:

```
; Program example, Wednesday, September 4, 2013
; STK600, CPU ATmega2560
; Addresses, see page 415 in doc2549_ATmega2560.pdf

ldi r16, 0b11111111
out 0x01, r16                 ; All one's to DDRA, outputs


ldi r16, 0b00000000
out 0x04, r16                 ; All one's to DDRB, inputs

my_loop:
in r16, 0x03                  ; read PINB
com r16                       ; one's complement each bit
out 0x02, r16                 ; write in PORTA
rjmp my_loop
```



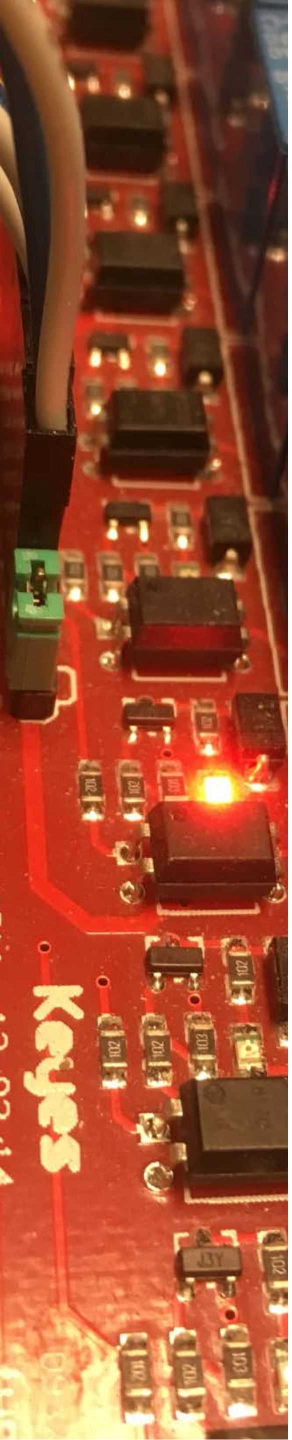
1ED022, Computer Technology

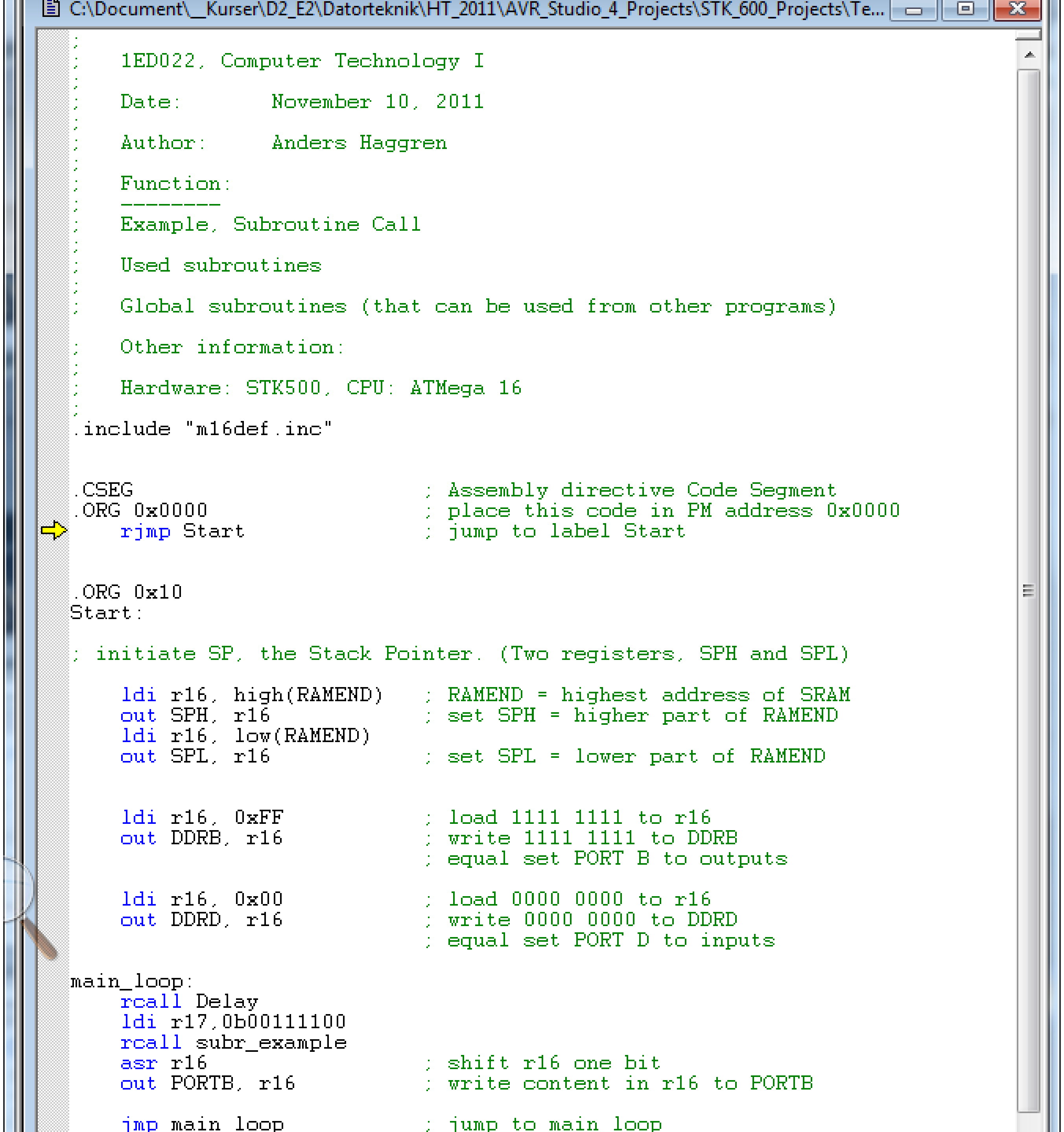
- Programming in Assembly
- STK500 or STK600
- Lectures
- Laboratory work (6 labs)
- Laboratory work is mandatory
- Written exam in January



Assembler commands

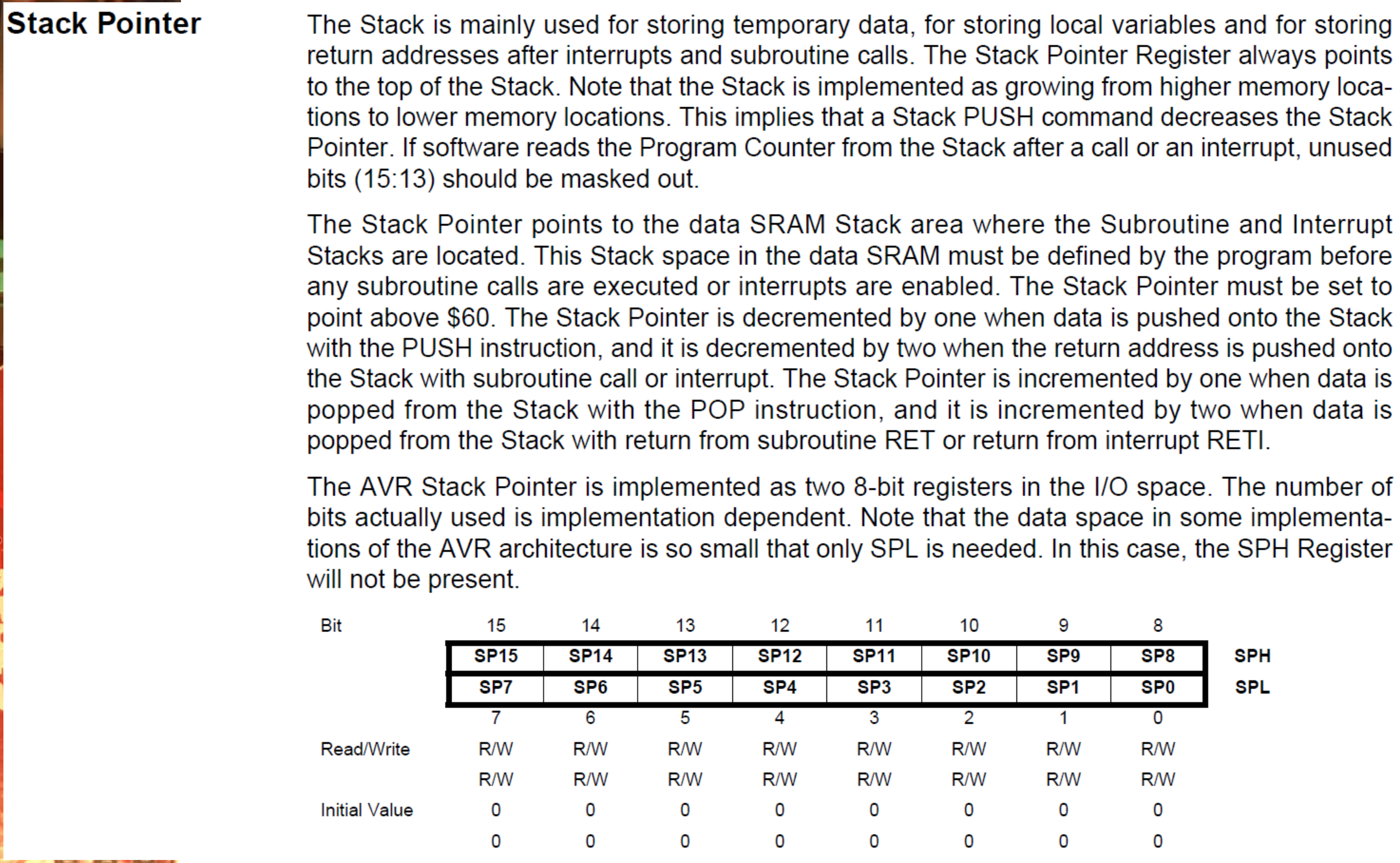
.EQU	Table_size	= 40
.DEF	my_reg	= r16
.ORG	0x10	
.CSEG		
.DSEG		
.BYTE		
.DW		
.INCLUDE		




C:\Document\__Kurser\D2_E2\Datorteknik\HT_2011\AVR_Studio_4_Projects\STK_600_Projects\Te...

```
;
;   1ED022, Computer Technology I
;
;   Date:       November 10, 2011
;
;   Author:     Anders Haggren
;
;   Function:
;   --------
;   Example, Subroutine Call
;
;   Used subroutines
;
;   Global subroutines (that can be used from other programs)
;
;   Other information:
;
;   Hardware: STK500, CPU: ATMega 16
;
.include "m16def.inc"


.CSEG                           ; Assembly directive Code Segment
.ORG 0x0000                     ; place this code in PM address 0x0000
    rjmp Start                  ; jump to label Start


.ORG 0x10
Start:

; initiate SP, the Stack Pointer. (Two registers, SPH and SPL)

    ldi r16, high(RAMEND)       ; RAMEND = highest address of SRAM
    out SPH, r16                ; set SPH = higher part of RAMEND
    ldi r16, low(RAMEND)
    out SPL, r16                ; set SPL = lower part of RAMEND


    ldi r16, 0xFF               ; load 1111 1111 to r16
    out DDRB, r16               ; write 1111 1111 to DDRB
                                ; equal set PORT B to outputs

    ldi r16, 0x00               ; load 0000 0000 to r16
    out DDRD, r16               ; write 0000 0000 to DDRD
                                ; equal set PORT D to inputs

main_loop:
    rcall Delay
    ldi r17,0b00111100
    rcall subr_example
    asr r16                     ; shift r16 one bit
    out PORTB, r16              ; write content in r16 to PORTB

    jmp main_loop               ; jump to main loop
```

Stack Pointer



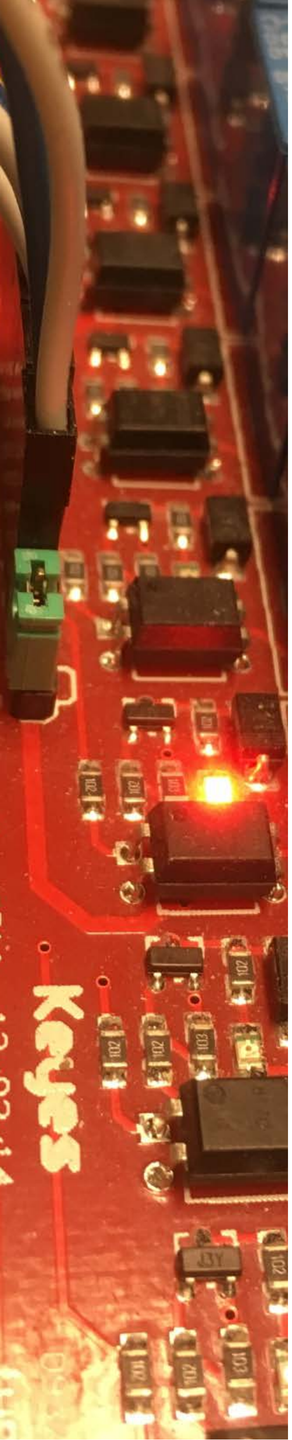
Stack Pointer

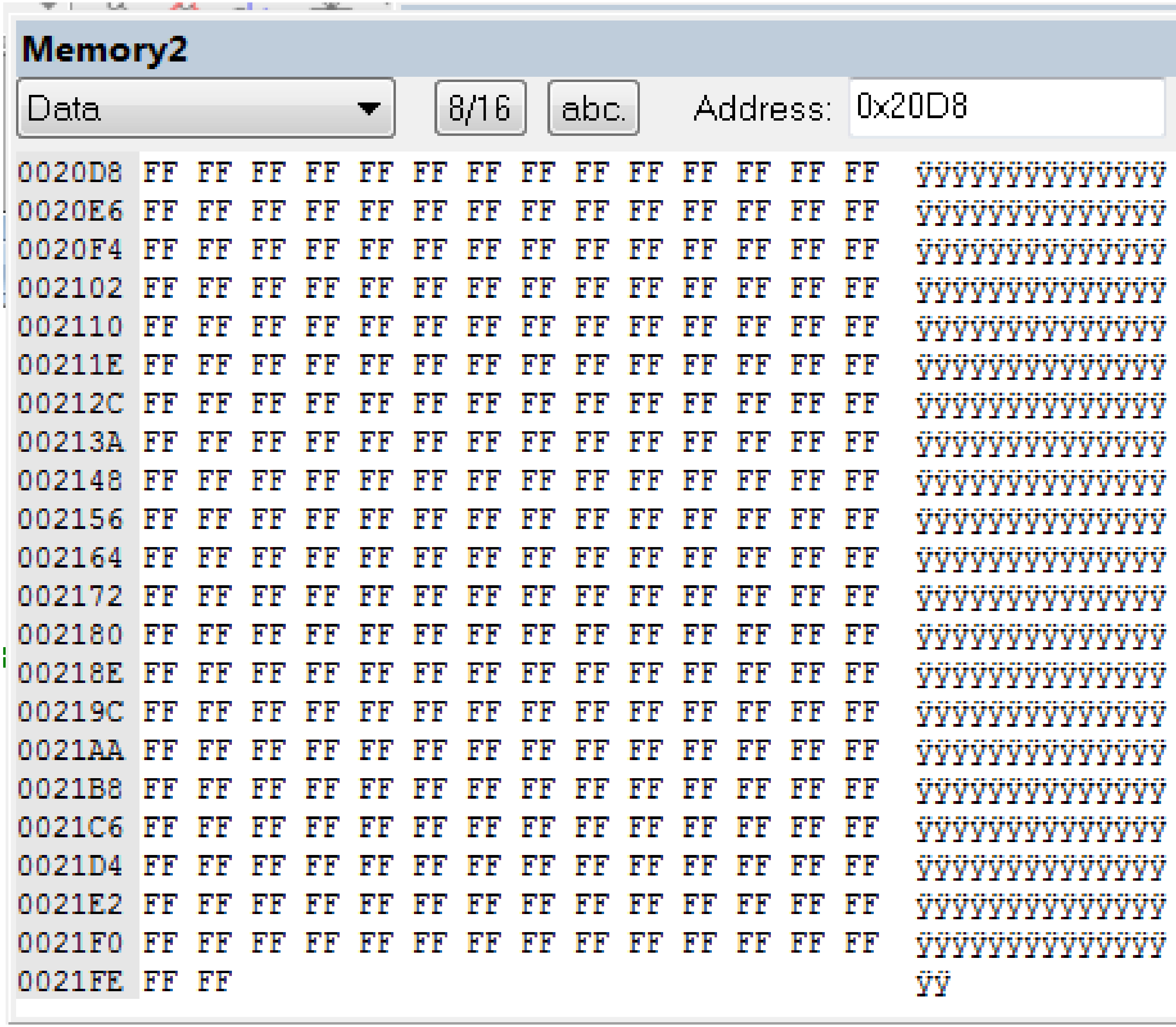
The Stack is mainly used for storing temporary data, for storing local variables and for storing return addresses after interrupts and subroutine calls. The Stack Pointer Register always points to the top of the Stack. Note that the Stack is implemented as growing from higher memory locations to lower memory locations. This implies that a Stack PUSH command decreases the Stack Pointer. If software reads the Program Counter from the Stack after a call or an interrupt, unused bits (15:13) should be masked out.

The Stack Pointer points to the data SRAM Stack area where the Subroutine and Interrupt Stacks are located. This Stack space in the data SRAM must be defined by the program before any subroutine calls are executed or interrupts are enabled. The Stack Pointer must be set to point above $60. The Stack Pointer is decremented by one when data is pushed onto the Stack with the PUSH instruction, and it is decremented by two when the return address is pushed onto the Stack with subroutine call or interrupt. The Stack Pointer is incremented by one when data is popped from the Stack with the POP instruction, and it is incremented by two when data is popped from the Stack with return from subroutine RET or return from interrupt RETI.

The AVR Stack Pointer is implemented as two 8-bit registers in the I/O space. The number of bits actually used is implementation dependent. Note that the data space in some implementations of the AVR architecture is so small that only SPL is needed. In this case, the SPH Register will not be present.

Bit	15	14	13	12	11	10	9	8	
	SP15	**SP14**	**SP13**	**SP12**	**SP11**	**SP10**	**SP9**	**SP8**	**SPH**
	SP7	**SP6**	**SP5**	**SP4**	**SP3**	**SP2**	**SP1**	**SP0**	**SPL**
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

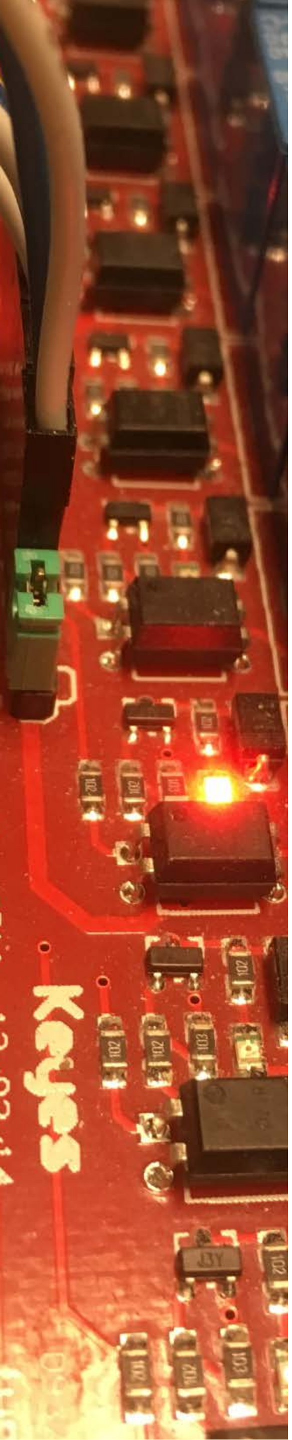




Memory2

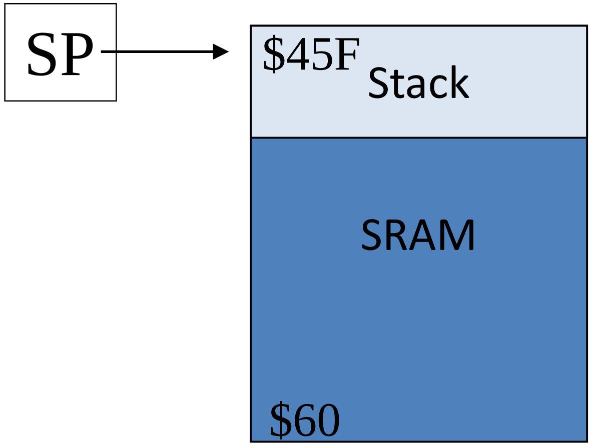
Address: 0x20D8

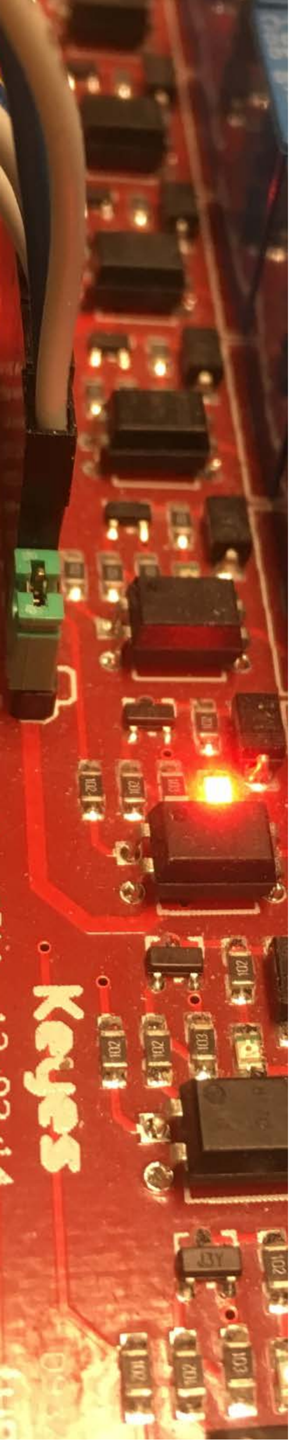
```
0020D8 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020E6 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020F4 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002102 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002110 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00211E FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00212C FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00213A FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002148 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002156 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002164 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002172 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002180 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00218E FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00219C FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021AA FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021B8 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021C6 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021D4 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021E2 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021F0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021FE FF FF                                      ÿÿ
```



Initiate the Stack Pointer, SP

- Typically, the stack starts at the highest SRAM address and grows downward through memory
- SP points to the next available byte of stack storage (top of stack after a push)



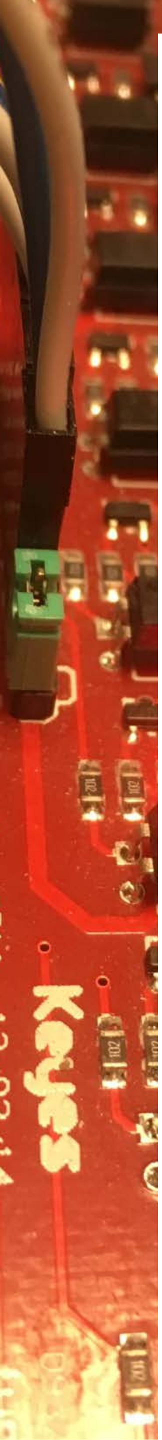


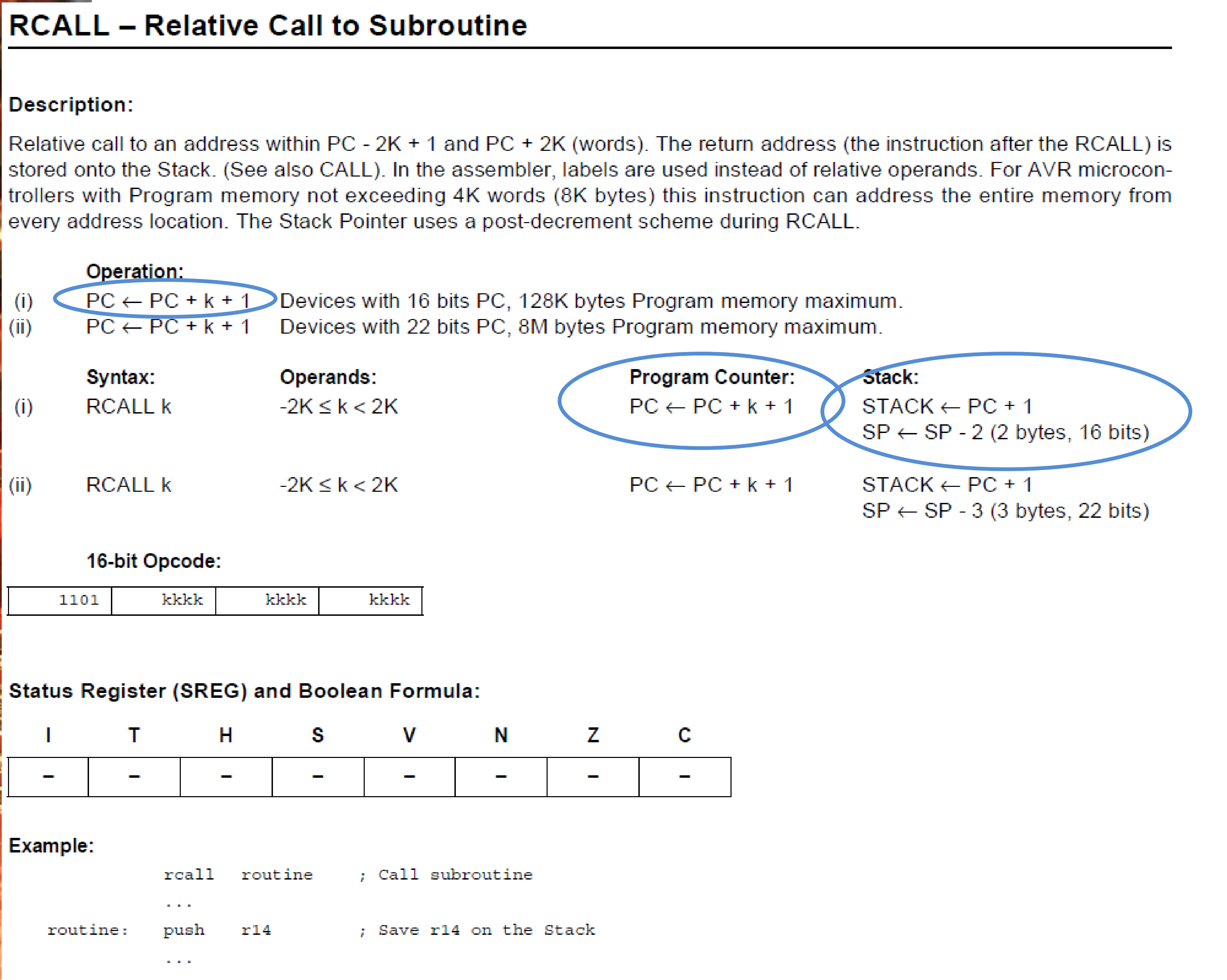
Initiate the Stack Pointer, SP

```
; initiate SP, the Stack Pointer. (Two registers, SPH and SPL)

ldi r16, high(RAMEND)        ; RAMEND = highest address of
SRAM
out SPH, r16                 ; set SPH = higher part of RAMEND
ldi r16, low(RAMEND)
out SPL, r16                 ; set SPL = lower part of RAMEN
```

RAMEND, SPH and SPL are defined in m16DEF.inc

```
;***** Specify Device
.device ATmega16

;***** I/O Register Definitions
.equu       SREG        =$3f
.equ        SPH         =$3e
.equ        SPL         =$3d
.equ        OCR0        =$3c
.equ        GICR        =$3b                    ; New name for GIMSK
```





RCALL – Relative Call to Subroutine

Description:

Relative call to an address within PC - 2K + 1 and PC + 2K (words). The return address (the instruction after the RCALL) is stored onto the Stack. (See also CALL). In the assembler, labels are used instead of relative operands. For AVR microcontrollers with Program memory not exceeding 4K words (8K bytes) this instruction can address the entire memory from every address location. The Stack Pointer uses a post-decrement scheme during RCALL.

	Operation:	
(i)	PC ← PC + k + 1	Devices with 16 bits PC, 128K bytes Program memory maximum.
(ii)	PC ← PC + k + 1	Devices with 22 bits PC, 8M bytes Program memory maximum.

	Syntax:	Operands:	Program Counter:	Stack:

(i)	RCALL k	-2K ≤ k < 2K	PC ← PC + k + 1	STACK ← PC + 1
				SP ← SP - 2 (2 bytes, 16 bits)

(ii)	RCALL k	-2K ≤ k < 2K	PC ← PC + k + 1	STACK ← PC + 1
				SP ← SP - 3 (3 bytes, 22 bits)

16-bit Opcode:

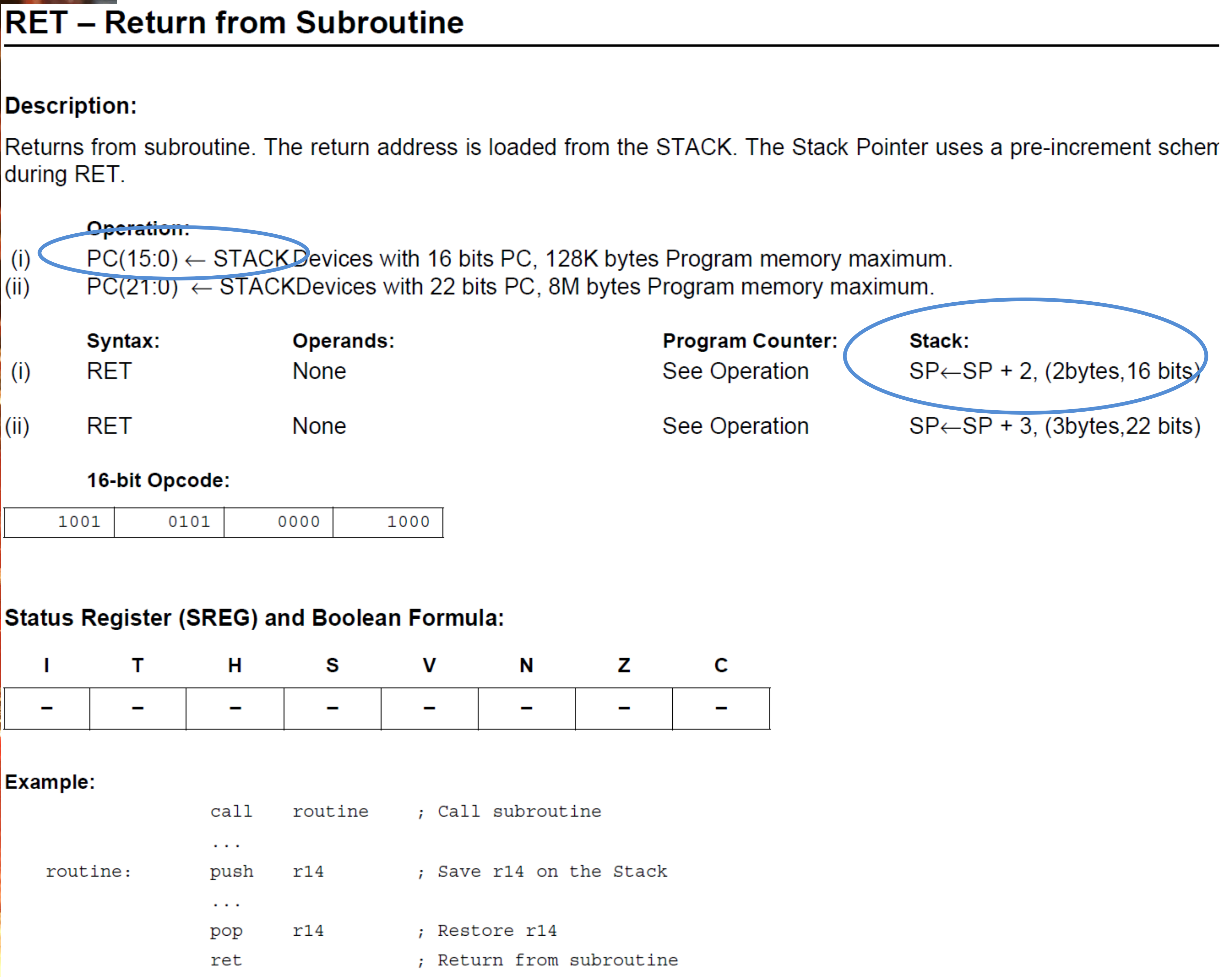
1101	kkkk	kkkk	kkkk

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
            rcall   routine     ; Call subroutine
            ...
   routine: push    r14         ; Save r14 on the Stack
            ...
```



RET – Return from Subroutine

Description:

Returns from subroutine. The return address is loaded from the STACK. The Stack Pointer uses a pre-increment schem during RET.

Operation:

(i) PC(15:0) ← STACK Devices with 16 bits PC, 128K bytes Program memory maximum.
(ii) PC(21:0) ← STACKDevices with 22 bits PC, 8M bytes Program memory maximum.

	Syntax:	Operands:	Program Counter:	Stack:
(i)	RET	None	See Operation	SP←SP + 2, (2bytes,16 bits)
(ii)	RET	None	See Operation	SP←SP + 3, (3bytes,22 bits)

16-bit Opcode:

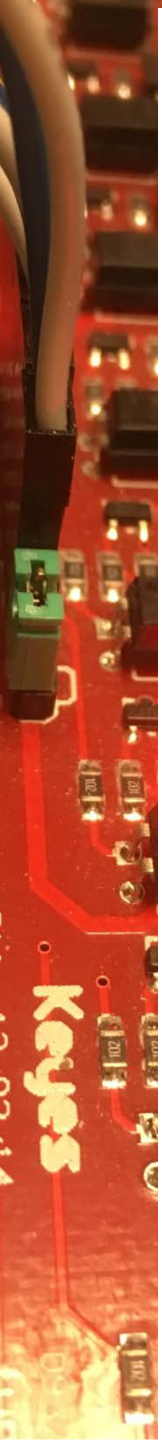
1001	0101	0000	1000

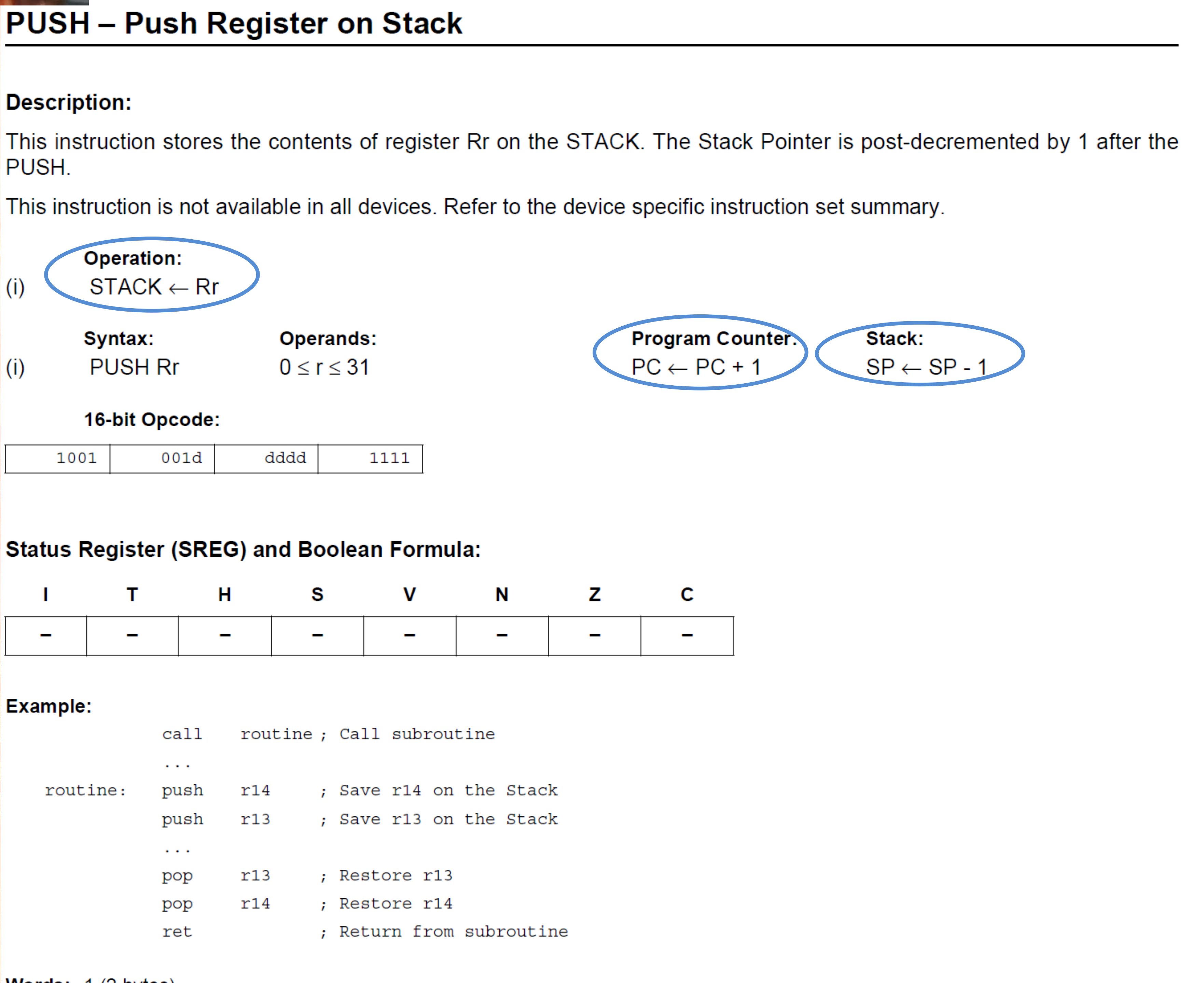
Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
             call    routine    ; Call subroutine
             ...
routine:     push    r14        ; Save r14 on the Stack
             ...
             pop     r14        ; Restore r14
             ret                ; Return from subroutine
```





PUSH – Push Register on Stack

Description:

This instruction stores the contents of register Rr on the STACK. The Stack Pointer is post-decremented by 1 after the PUSH.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) STACK ← Rr

	Syntax:	Operands:	Program Counter:	Stack:
(i)	PUSH Rr	0 ≤ r ≤ 31	PC ← PC + 1	SP ← SP - 1

16-bit Opcode:

1001	001d	dddd	1111

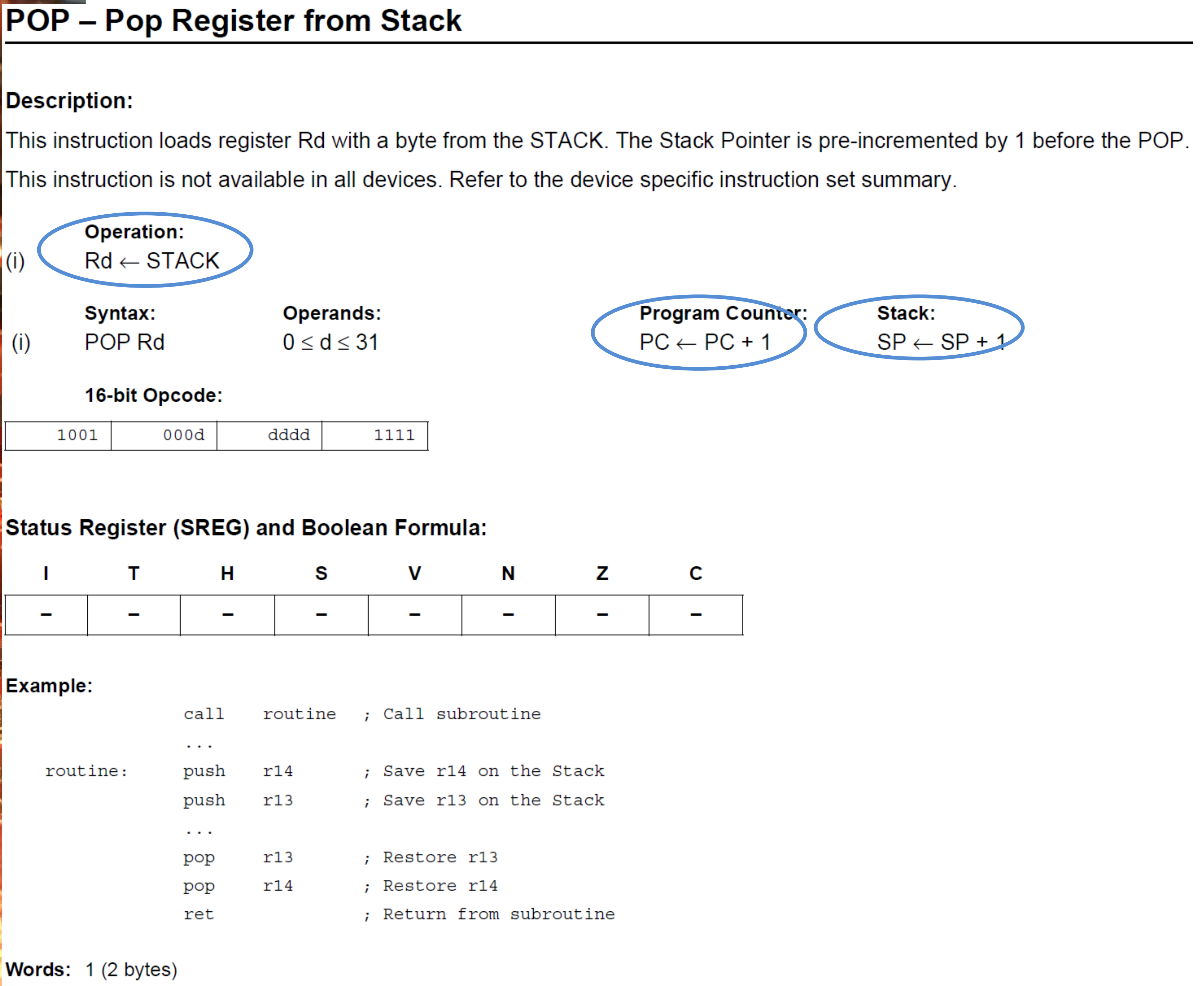
Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
            call    routine ; Call subroutine
            ...
routine:    push    r14     ; Save r14 on the Stack
            push    r13     ; Save r13 on the Stack
            ...
            pop     r13     ; Restore r13
            pop     r14     ; Restore r14
            ret             ; Return from subroutine
```

Words: 1 (2 bytes)



POP – Pop Register from Stack

Description:

This instruction loads register Rd with a byte from the STACK. The Stack Pointer is pre-incremented by 1 before the POP.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) Rd ← STACK

	Syntax:	Operands:	Program Counter:	Stack:
(i)	POP Rd	$0 \le d \le 31$	PC ← PC + 1	SP ← SP + 1

16-bit Opcode:

1001	000d	dddd	1111

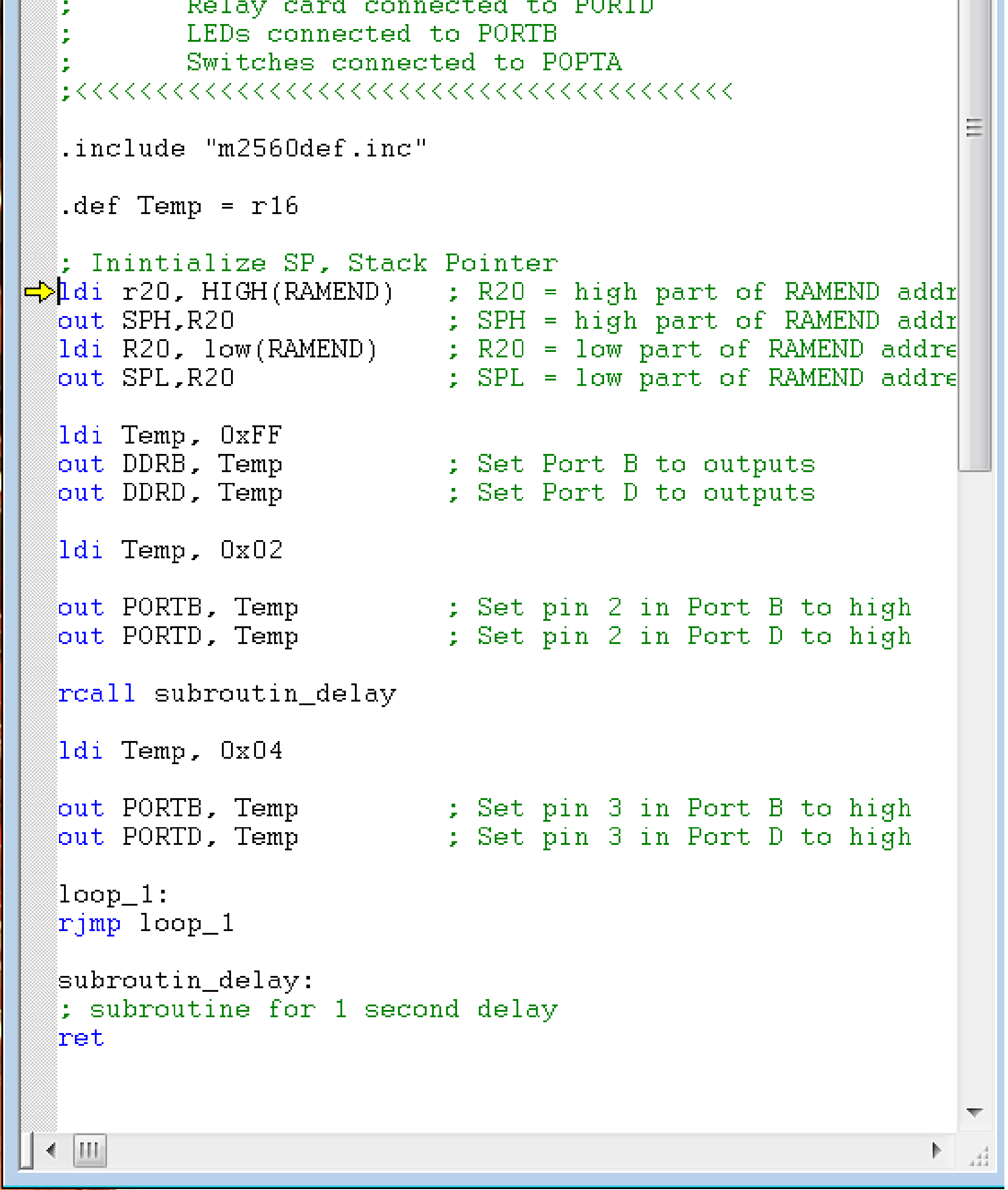
Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

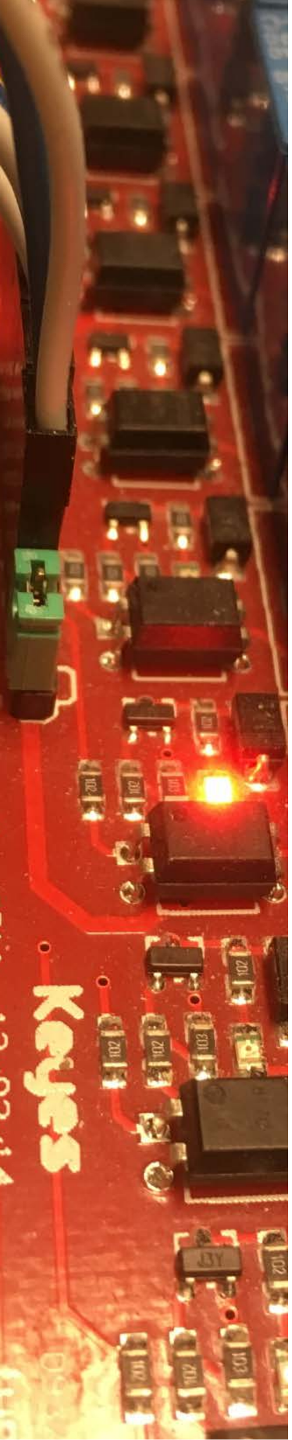
Example:

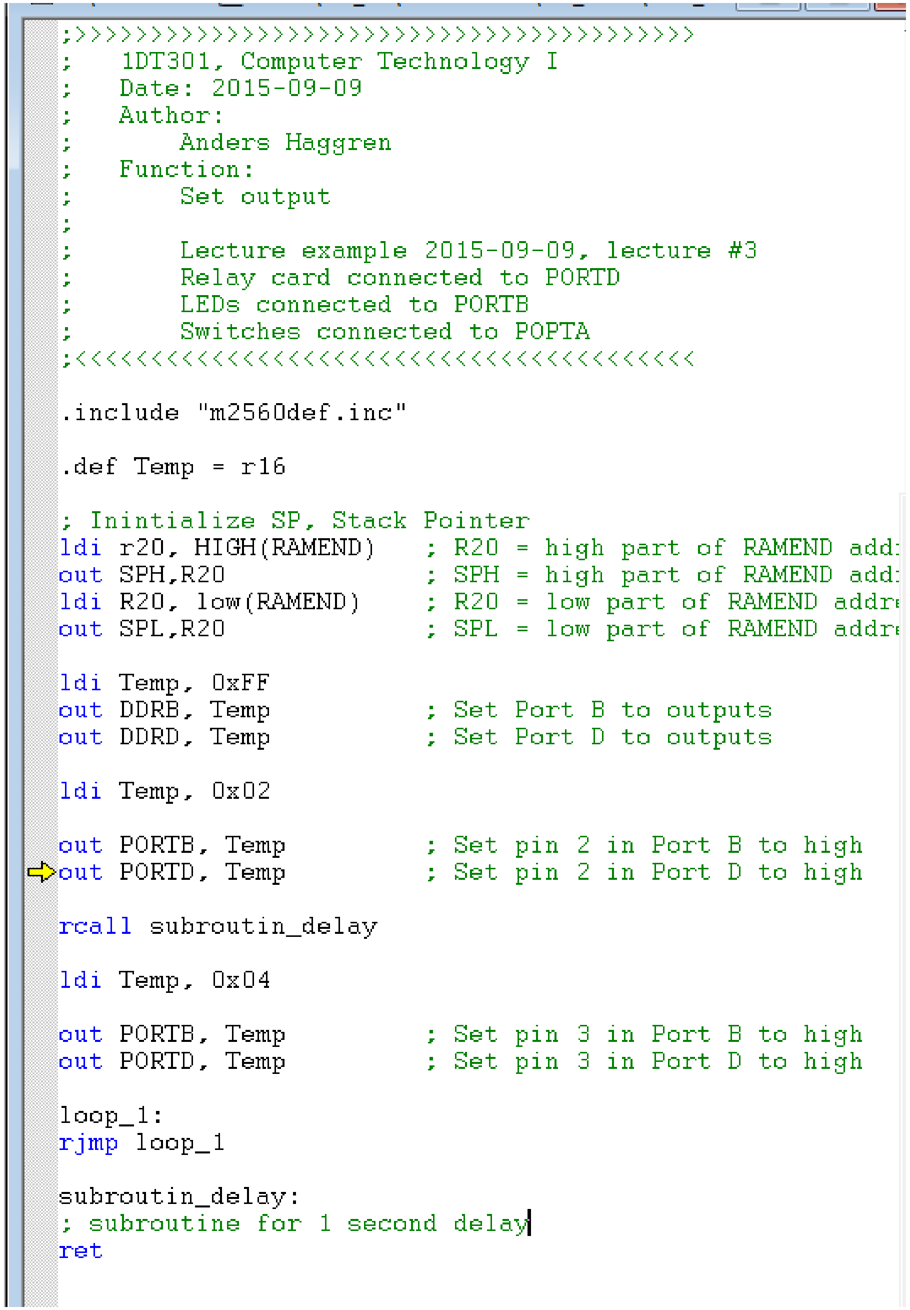
```
                call    routine    ; Call subroutine
                ...
    routine:    push    r14        ; Save r14 on the Stack
                push    r13        ; Save r13 on the Stack
                ...
                pop     r13        ; Restore r13
                pop     r14        ; Restore r14
                ret                ; Return from subroutine
```

Words: 1 (2 bytes)

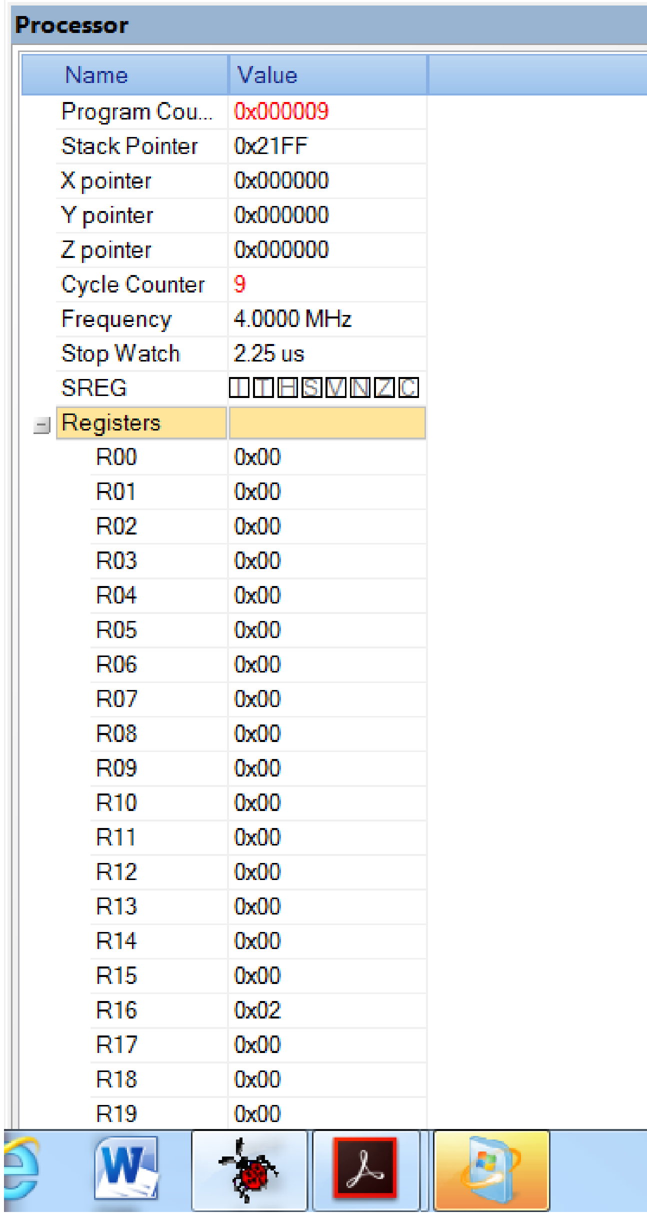


```
;       Relay card connected to PORTD
;       LEDs connected to PORTB
;       Switches connected to POPTA
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

.include "m2560def.inc"

.def Temp = r16

; Inintialize SP, Stack Pointer
ldi r20, HIGH(RAMEND)   ; R20 = high part of RAMEND addr
out SPH,R20             ; SPH = high part of RAMEND addr
ldi R20, low(RAMEND)    ; R20 = low part of RAMEND addre
out SPL,R20             ; SPL = low part of RAMEND addre

ldi Temp, 0xFF
out DDRB, Temp          ; Set Port B to outputs
out DDRD, Temp          ; Set Port D to outputs

ldi Temp, 0x02

out PORTB, Temp         ; Set pin 2 in Port B to high
out PORTD, Temp         ; Set pin 2 in Port D to high

rcall subroutin_delay

ldi Temp, 0x04

out PORTB, Temp         ; Set pin 3 in Port B to high
out PORTD, Temp         ; Set pin 3 in Port D to high

loop_1:
rjmp loop_1

subroutin_delay:
; subroutine for 1 second delay
ret
```



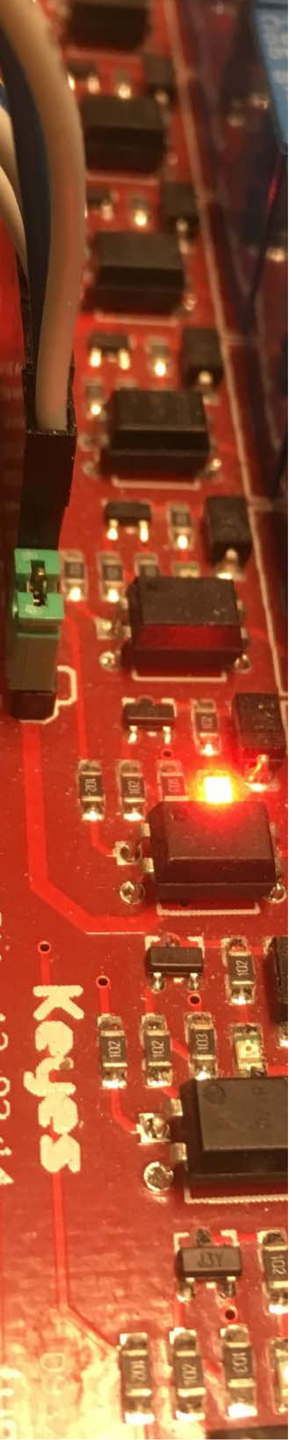


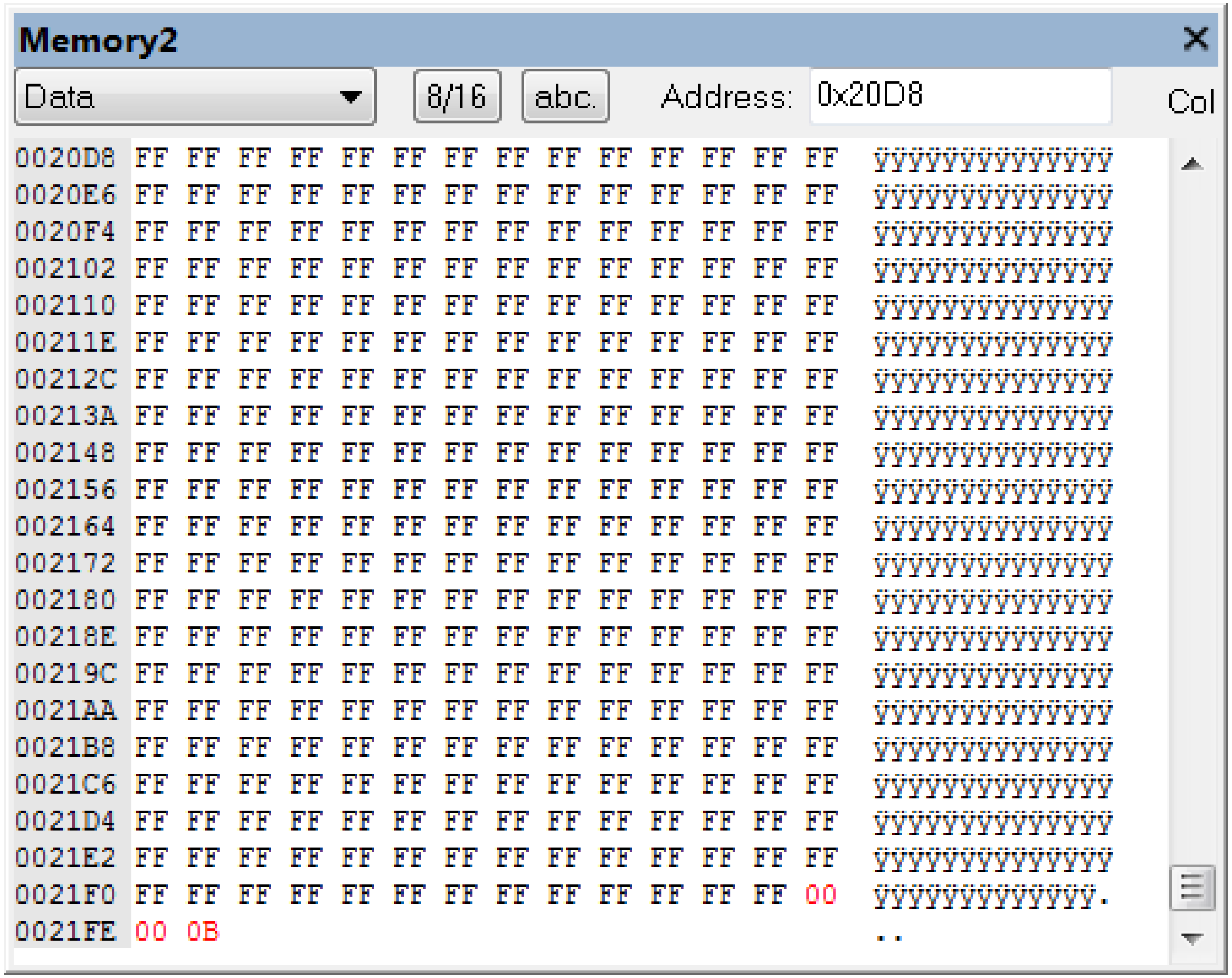
```
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;   1DT301, Computer Technology I
;   Date: 2015-09-09
;   Author:
;       Anders Haggren
;   Function:
;       Set output
;
;       Lecture example 2015-09-09, lecture #3
;       Relay card connected to PORTD
;       LEDs connected to PORTB
;       Switches connected to POPTA
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

.include "m2560def.inc"

.def Temp = r16

; Inintialize SP, Stack Pointer
ldi r20, HIGH(RAMEND)    ; R20 = high part of RAMEND add:
out SPH,R20              ; SPH = high part of RAMEND add:
ldi R20, low(RAMEND)     ; R20 = low part of RAMEND addre
out SPL,R20              ; SPL = low part of RAMEND addre

ldi Temp, 0xFF
out DDRB, Temp           ; Set Port B to outputs
out DDRD, Temp           ; Set Port D to outputs

ldi Temp, 0x02

out PORTB, Temp          ; Set pin 2 in Port B to high
out PORTD, Temp          ; Set pin 2 in Port D to high

rcall subroutin_delay

ldi Temp, 0x04

out PORTB, Temp          ; Set pin 3 in Port B to high
out PORTD, Temp          ; Set pin 3 in Port D to high

loop_1:
rjmp loop_1

subroutin_delay:
; subroutine for 1 second delay
ret
```



Processor

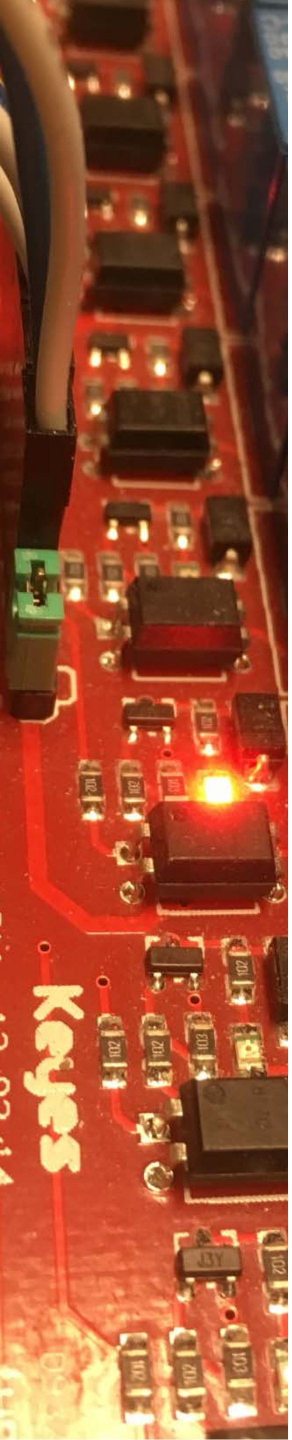
Name	Value
Program Cou...	0x000009
Stack Pointer	0x21FF
X pointer	0x000000
Y pointer	0x000000
Z pointer	0x000000
Cycle Counter	9
Frequency	4.0000 MHz
Stop Watch	2.25 us
SREG	I T H S V N Z C
Registers	
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x02
R17	0x00
R18	0x00
R19	0x00

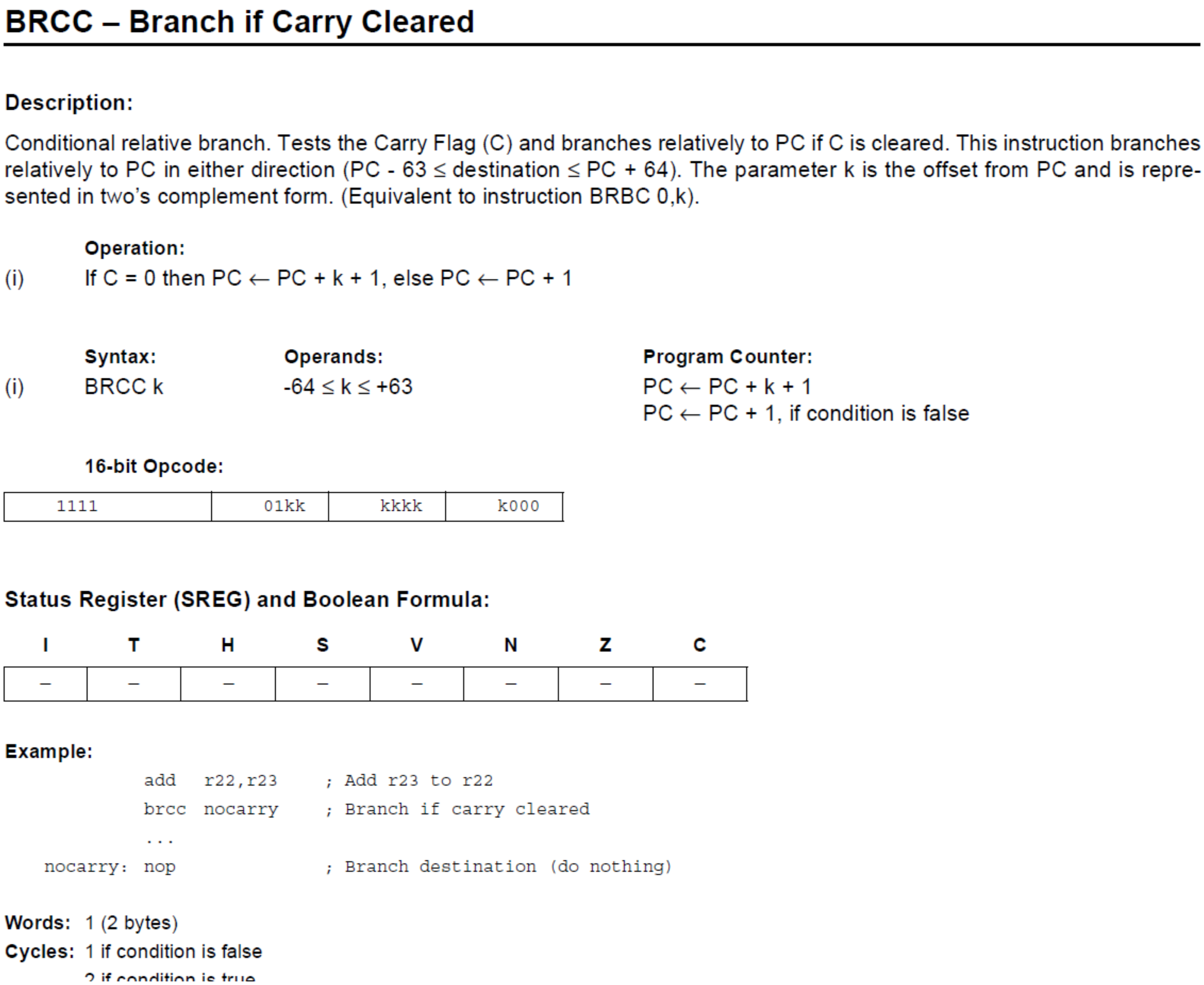




Address: 0x20D8

```
0020D8 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020E6 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020F4 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002102 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002110 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00211E FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00212C FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00213A FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002148 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002156 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002164 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002172 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
002180 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00218E FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00219C FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021AA FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021B8 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021C6 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021D4 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021E2 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0021F0 FF FF FF FF FF FF FF FF FF FF FF FF FF 00  ÿÿÿÿÿÿÿÿÿÿÿÿÿ.
0021FE 00 0B                                      ..
```





BRCC – Branch if Carry Cleared

Description:

Conditional relative branch. Tests the Carry Flag (C) and branches relatively to PC if C is cleared. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 0,k).

	Operation:
(i)	If C = 0 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	**Operands:**	**Program Counter:**
(i)	BRCC k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	01kk	kkkk	k000

Status Register (SREG) and Boolean Formula:

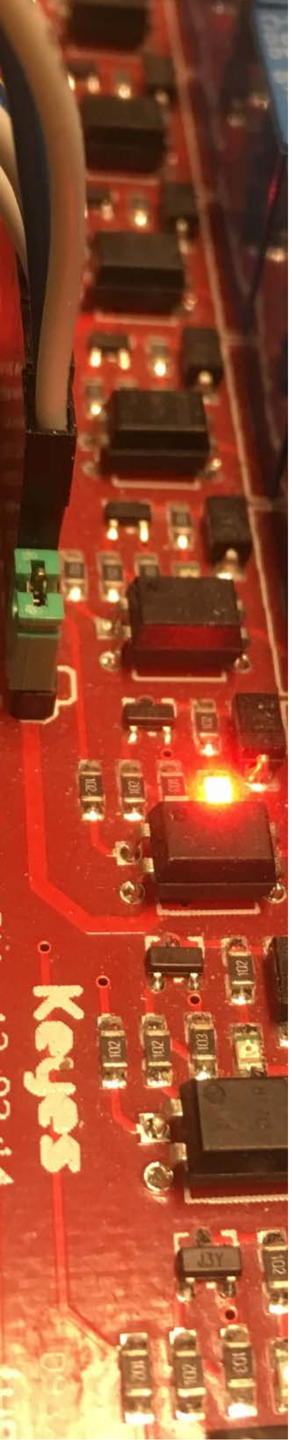
I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

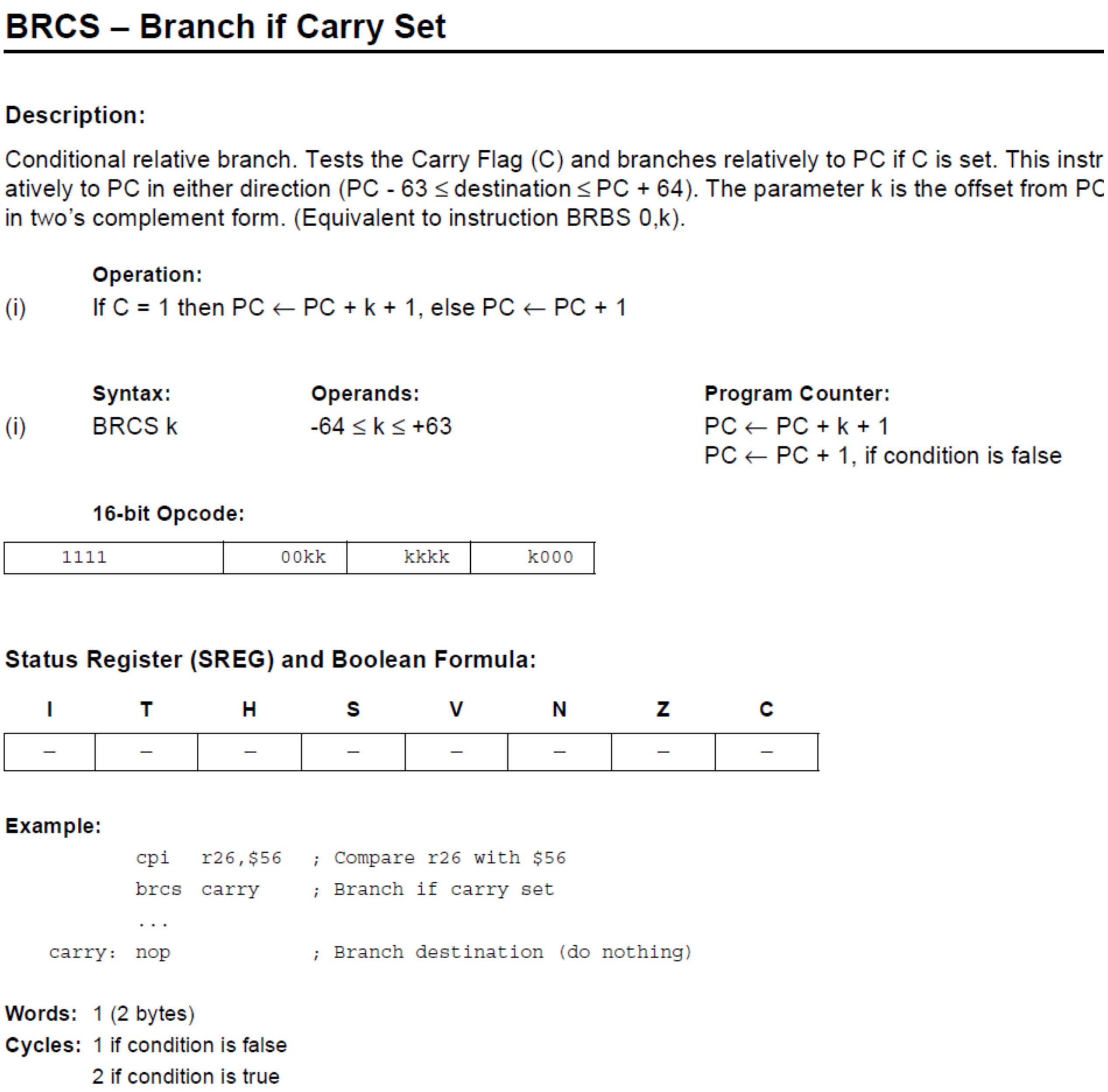
Example:

```
          add   r22,r23     ; Add r23 to r22
          brcc  nocarry     ; Branch if carry cleared
          ...
nocarry:  nop               ; Branch destination (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true





BRCS – Branch if Carry Set

Description:

Conditional relative branch. Tests the Carry Flag (C) and branches relatively to PC if C is set. This instr
atively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC
in two's complement form. (Equivalent to instruction BRBS 0,k).

Operation:

(i) If C = 1 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRCS k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	00kk	kkkk	k000

Status Register (SREG) and Boolean Formula:

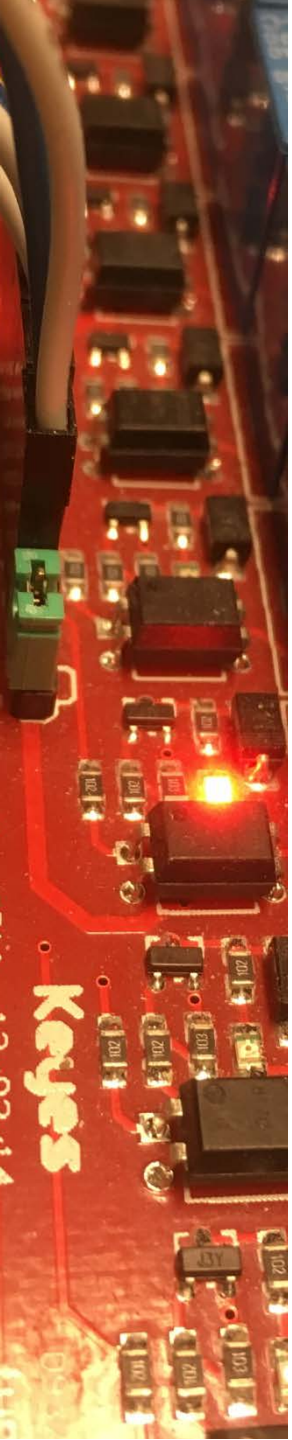
I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
        cpi   r26,$56    ; Compare r26 with $56
        brcs  carry      ; Branch if carry set
        ...
carry:  nop              ; Branch destination (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true



BREQ – Branch if Equal

Description:

Conditional relative branch. Tests the Zero Flag (Z) and branches relatively to PC if Z is set. If the instruction is executed immediately after any of the instructions CP, CPI, SUB or SUBI, the branch will occur if and only if the unsigned or signed binary number represented in Rd was equal to the unsigned or signed binary number represented in Rr. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 1,k).

Operation:

(i) If Rd = Rr (Z = 1) then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BREQ k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

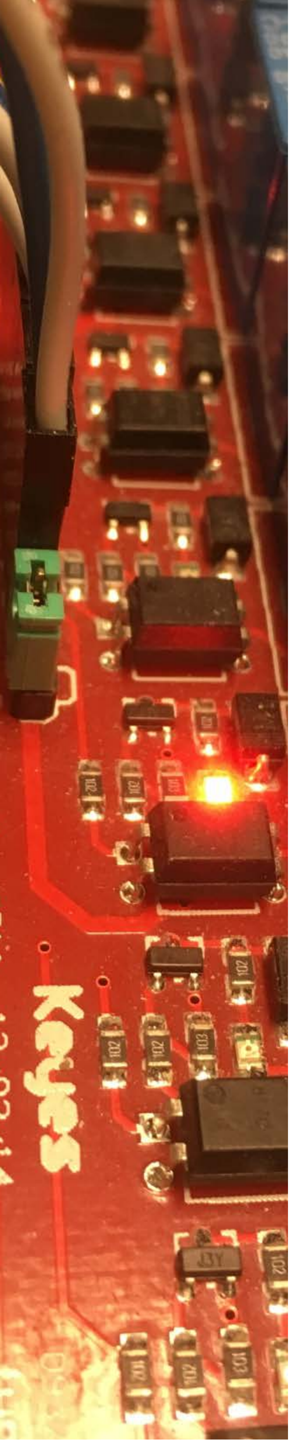
1111	00kk	kkkk	k001

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
        cp    r1,r0     ; Compare registers r1 and r0
        breq  equal     ; Branch if registers equal
        ...
equal:  nop             ; Branch destination (do nothing)
```



BRGE – Branch if Greater or Equal (Signed)

Description:

Conditional relative branch. Tests the Signed Flag (S) and branches relatively to PC if S is cleared. If the instruction is executed immediately after any of the instructions CP, CPI, SUB or SUBI, the branch will occur if and only if the signed binary number represented in Rd was greater than or equal to the signed binary number represented in Rr. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 4,k).

Operation:

(i) If Rd ≥ Rr (N ⊕ V = 0) then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRGE k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

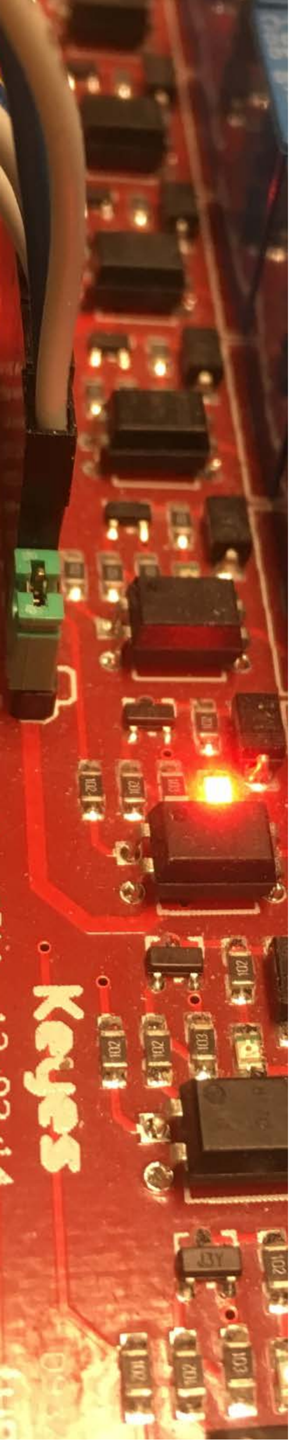
1111	01kk	kkkk	k100

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

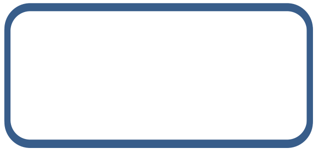
```
            cp    r11,r12     ; Compare registers r11 and r12
            brge  greateq     ; Branch if r11 ≥ r12 (signed)
            ...
   greateq: nop               ; Branch destination (do nothing)
```



Flowcharts Building Blocks:

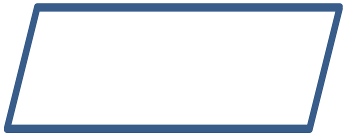
Rounded Rectangle

Used for: Start process, End process



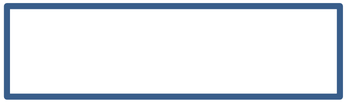
Parallelogram

Used for: Input, Output



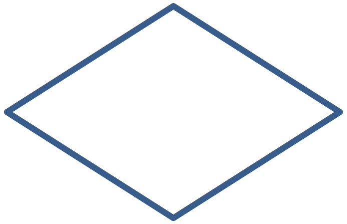
Rectangle

Used for: Processing, Calculations



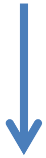
Rhombus

Used for: Decision Making (Yes/No branching, IF/ELSE)

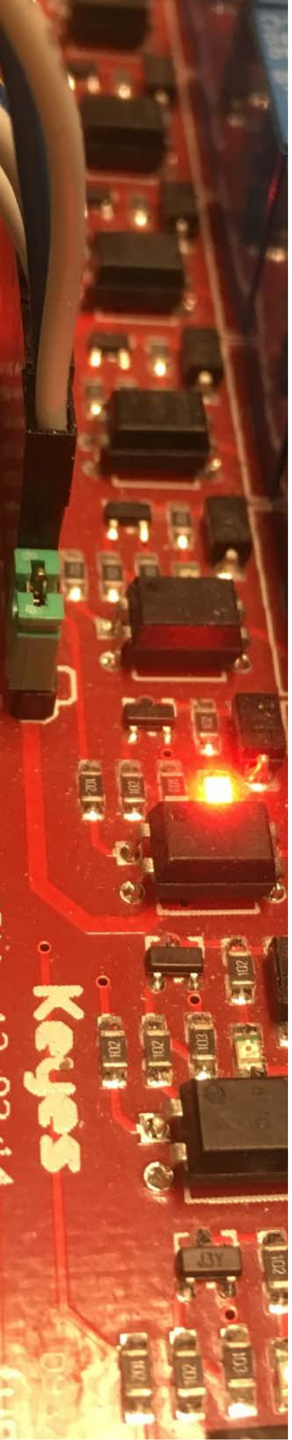


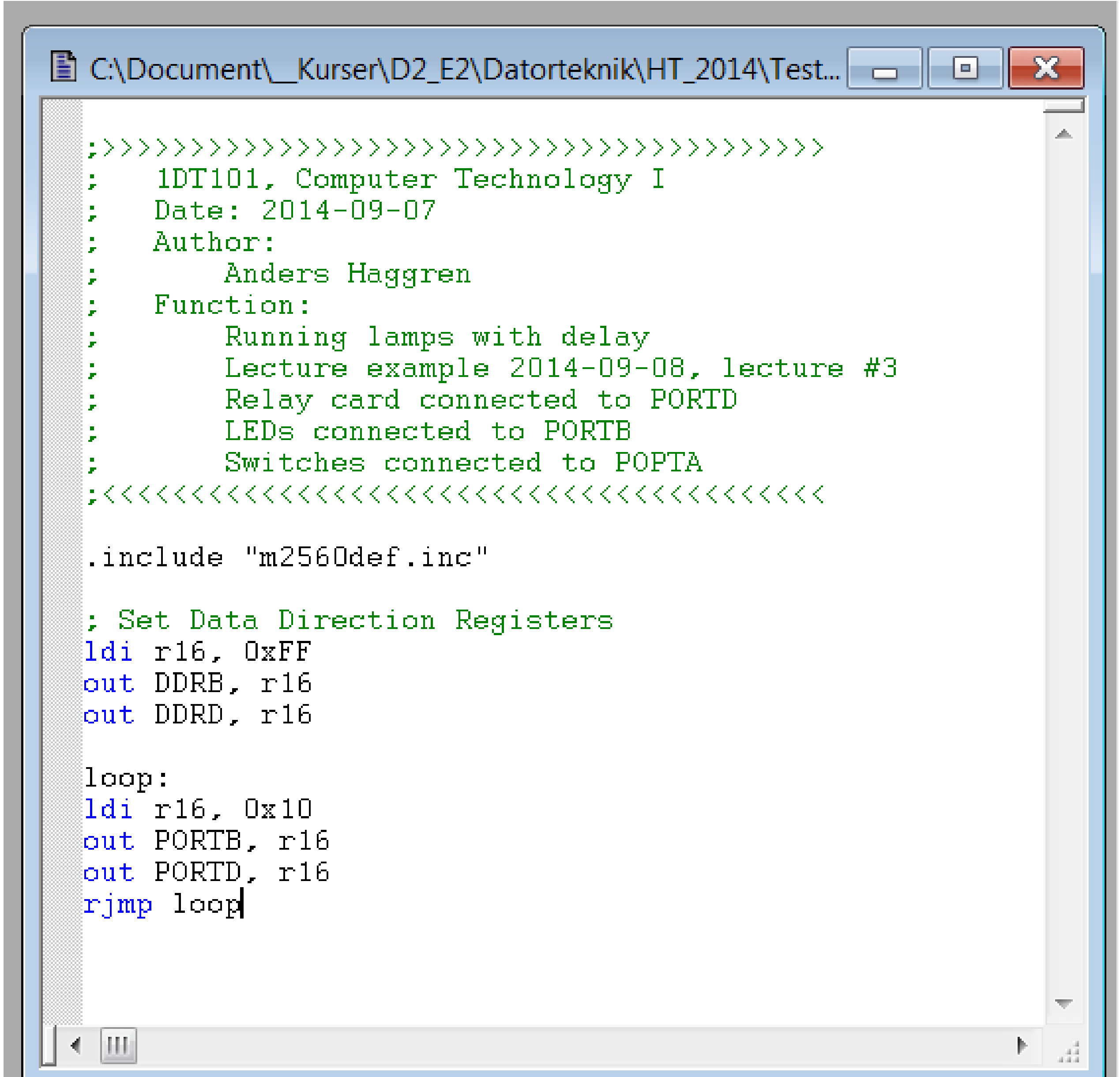
Arrow

Used for: Flow/direction of the algorithm steps

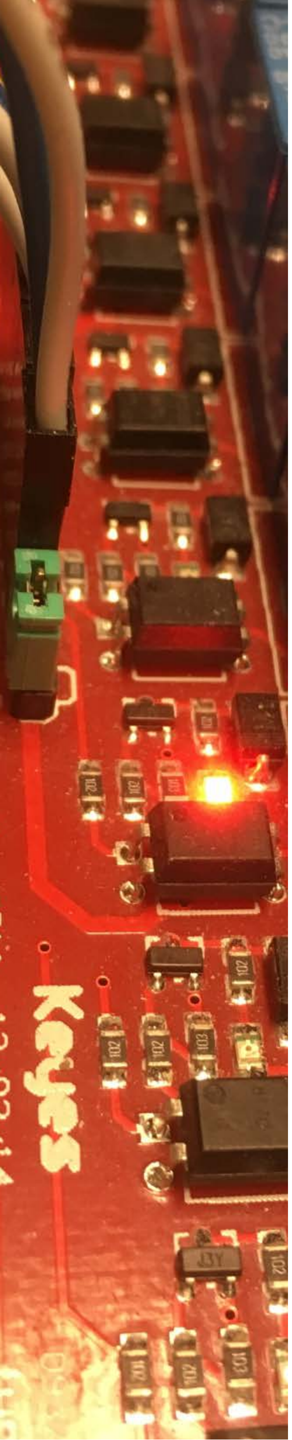


Program example:

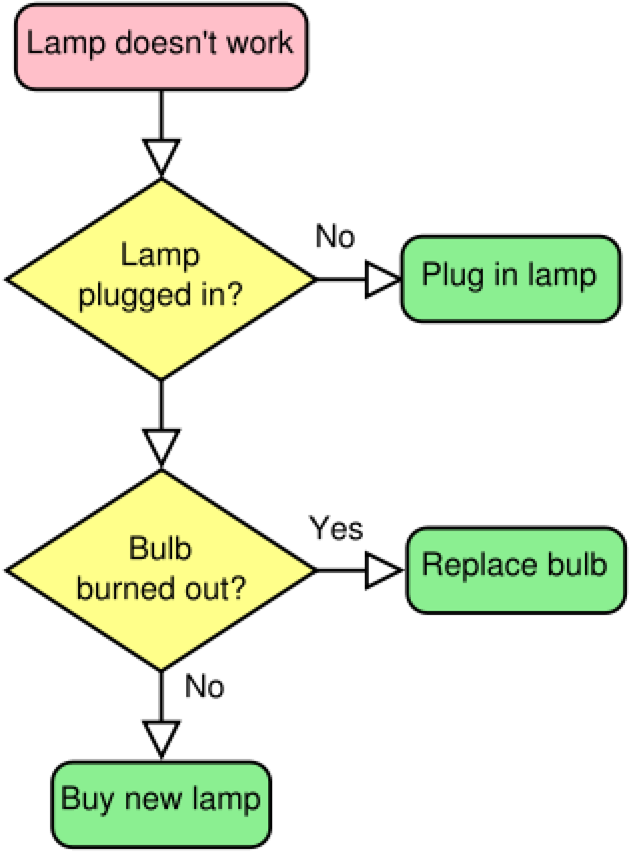


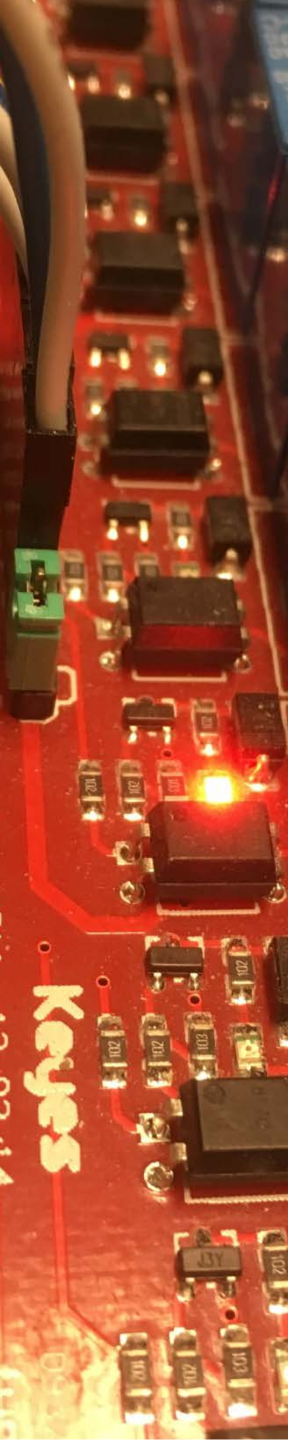


```
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;   1DT101, Computer Technology I
;   Date: 2014-09-07
;   Author:
;       Anders Haggren
;   Function:
;       Running lamps with delay
;       Lecture example 2014-09-08, lecture #3
;       Relay card connected to PORTD
;       LEDs connected to PORTB
;       Switches connected to POPTA
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

.include "m2560def.inc"

; Set Data Direction Registers
ldi r16, 0xFF
out DDRB, r16
out DDRD, r16

loop:
ldi r16, 0x10
out PORTB, r16
out PORTD, r16
rjmp loop
```

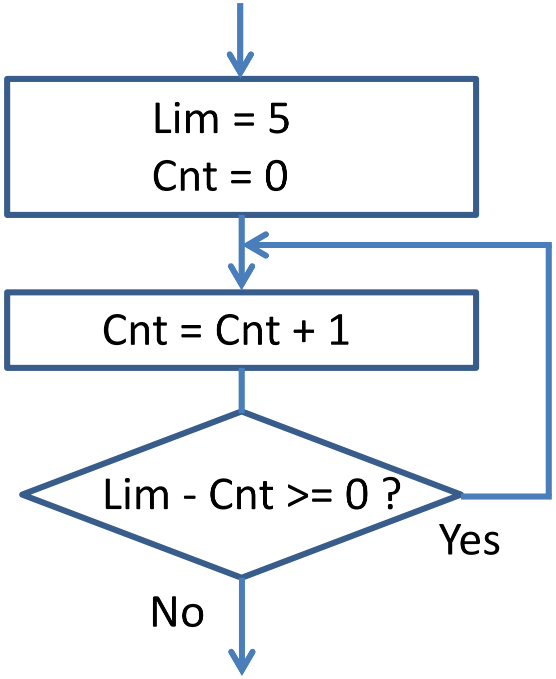


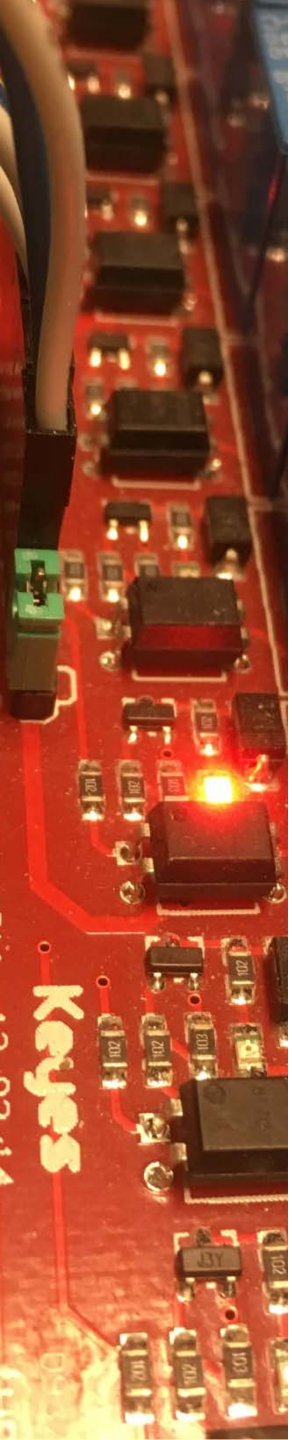
Using Flow Charts





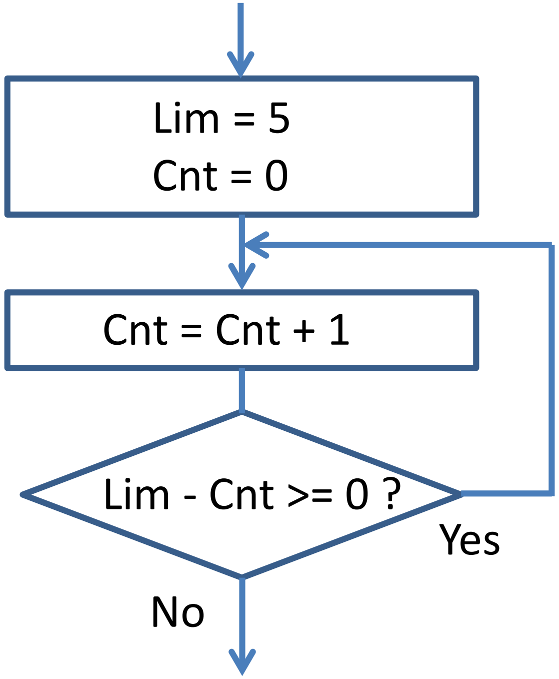
Loop:

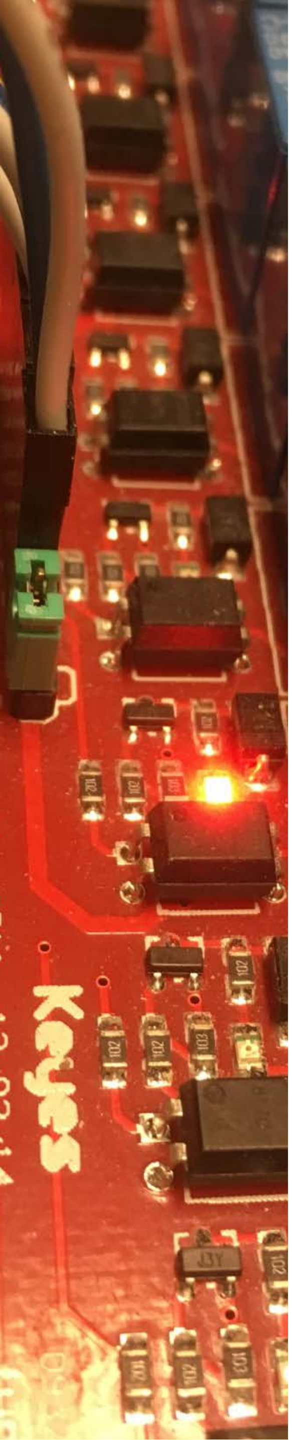


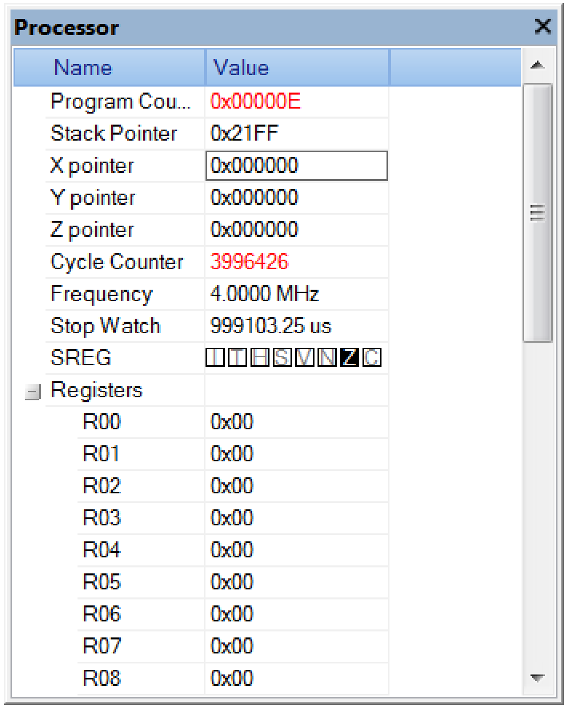


Loop:

```
Delay:

          ldi r16, 5              ; r16 = limit value
          ldi r17, 0              ; r17 = loop counter

del_1:
          inc r17                 ; r17 = r17 + 1
          cp r16, r17             ; compare r16 - 17
          brge del_1              ; if greater or equal, go back
          ret
```







Processor

Name	Value
Program Cou...	0x00000E
Stack Pointer	0x21FF
X pointer	0x000000
Y pointer	0x000000
Z pointer	0x000000
Cycle Counter	3996426
Frequency	4.0000 MHz
Stop Watch	999103.25 us
SREG	I T H S V N Z C
Registers	
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00