1DT301, Computer Technology I

Lecture #3,
Tuesday, September 10, 2019

- Subroutine call, rcall
- Use of Stack
- Push and Pop

Reading:

- http://www.avr-asm-tutorial.net/avr_en/
- http://www.avrbeginners.net/

AVR-Tutorial

www.avr-asm-tutorial.net/avr_en/ 67% Sök

Path: Home => AVR-Overview

Tutorial for learning assembly language for the
AVR-Single-Chip-Processors
(AT90S, ATmega and ATtiny) from ATMEL with practical examples.

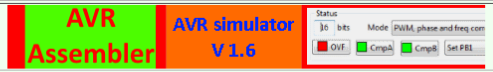
The Single-Chip-processors of ATMEL are excellent for homebrewing every kind of processor-driven electronics. The only problem is that assembly has to be learned in order to program these devices. After having done these first steps the assembly language provides very fast, lean and effective code, by which every task can be accomodated. These pages are for beginners and help in learning the first steps.

Sitemap | New on this webpage | Error list | AVR-Webring

Index

An overview on AVRs and their hardware

A simulator for AVR assembler code



Learning from applications

More than 60,000 downloads of assembler source code files in 2018

The four most popular applications:

Reading keypad keys
Frequency counter
Signal generator
Stepper motor driver ATtiny13

The four most recently added applications:

ATtiny and crystals to generate accurate rectangles
RGB BCD watch
DCF77 signal analysis
Large watch ATmega48

Use of controller internal hardware components in assembler source codes on this webseite

Software a

Beginners introduction to AVR

Beginner's introduction to AVR assembler language Also available as complete PDF document for printing the whole course (Download, 1.1 MB)

micro beginner course

A micro beginner course in 14 lectures from simple to complex with an ATtiny13 and an ATtiny24, hard- and software, all on a breadboard, also available in one complete PDF document

A starter course for beginners using the simulator to examine the effect of assembler instructions, also available as a PDF document

1010.1010
+101.0110
=???

An introduction to binary numbers and binary math with practical code examples for adding, subtracting, multiplying and dividing of binary numbers of any size

Four simple programming examples with extended comments as first steps of a practical introduction to assembler programming: Sense and requirements, Simple programming examples

?

Software-Know-How, special assembler commands: LPM, stack jumps, macros

All about timing loops from microseconds over milliseconds and seconds to hours, days and monthes: all you need is a loop, or two, or three ...

All you need to know about interrupts for beginners, on int vectors, int sources, on int execution and how ints govern the whole program design, a whole new world to learn about

Tools for programming in assembler

Source: http://www.avr-asm-tutorial.net/avr_en/

Beginners Programming in AVR Assembler

The following pages are written for all people that try to program in assembler language for the first time and like to start with programming ATMEL-AVRs AT90S, ATtiny, ATmega. Some basic programming techniques are shown that are necessary. During later learning stages these informations are useful and so try to learn this first. Tables are included for easier programming. There might be some bugs in these pages. Bug reports and hints for improving these pages are very welcome.

Overview on the beginners programming tutorial

Theme	What is that	Instructions
Hardware	ISP-Interface, Parallel-Port-Programmer, Experimental 2313 board, Commercial boards	-
Why	Why learn assembler language?, The concept behind assembler, Planning an AVR project	-
Tools	Editor, Assembler, ISP, Studio3, Studio4, Structure	-
Register	What is a Register?	.DEF, LDI, MOV
	Which different registers are available	CLR, ANDI, CBR, CPI, SBCI, SBR, SER, SUBI
	Register as Pointer	
	What can be packed into a register?	-
Ports	What is a Port?	.INCLU
	Which Ports are available?	-
	Status register as most relevant port	CLx, SEx, BCLR, BSET
SRAM	What is SRAM?	-
	For which purposes SRAM is used	-
	How to use SRAM?	STS, LDS, LD, ST, STD, LDD
	Stack in SRAM	PUSH, POP, RCALL, RET
Program run	What happens during a Reset?	-
	Linear program runs and branches	.CSEG, .DSEG, .ORG, .ESEG, INC, BRNE, BREQ, BRxx
	Timing of commands	NOP, DEC
	Macros	.MACRO, .ENDMACRO
	Subroutines	RET, RCALL, RJMP, SBRC, SBRS, SBIC, SBIS
	Interrupts	RETI
Calculations	Numbers and characters	-
	Bit manipulation	ORI, OR, ANDI, AND, CBR, SBR, EOR, COM, NEG, BLD, CLT, SET, BST
	Shift and rotate	LSL, LSR, ASR, ROL, ROR, SWAP
	Adding, Subtracting, Compares	ADD, ADC, SUB, SBC, CP, CPC, CPI, TST
	Format changes	.DB, .DW
Tables	Commands by function	-
	Commands alphabetic	-
	Ports	-
	Vectors	-
	Abbreviations	-
	Directives	all dot directives
	Expressions	-

Example: What is a Register?

To the top of this page

If you need a complete description of all instructions of the AVRs with explanations you might be happy to have the document here by Anthony Peck. It also provides small projects with an 8-pin ATtiny13.

Source: http://www.avr-asm-tutorial.net/avr_en/beginner/index.html

Introduction to AVR assembler programming for beginners

What is a register?

Registers are special storages with 8 bits capacity and they look like this:

7	6	5	4	3	2	1	0

Note the numeration of these bits: the least significant bit starts with zero ($2^0 = 1$).
A register can either store numbers from 0 to 255 (positive number, no negative values), or numbers from -128 to +127 (whole number with a sign bit in bit 7), or a value representing an ASCII-coded character (e.g. 'A'), or just eight single bits that do not have something to do with each other (e.g. for eigh single flags used to signal eight different yes/no decisions).
The special character of registers, compared to other storage sites, is that

- they can be used directly in assembler commands,
- operations with their content require only a single command word,
- they are connected directly to the central processing unit called the accumulator,
- they are source and target for calculations.

There are 32 registers in an AVR. They are originally named R0 to R31, but you can choose to name them to more meaningful names using an assemble directive. An example:

.DEF MyPreferredRegister = R16

Note that assembler directives like this are only meaningful for the assembler but do not produce any code that is executable in the AVR target chip. Instead of using the register name R16 we can now use our own name MyPreferredRegister, if we want to use R16 within a command. So we write a little bit more text each time we use this register, but we have an association what might be the content of this register.

Using the command line

Source: http://www.avr-asm-tutorial.net/avr_en/beginner/index.html

Beginners Programming in AVR Assembler

The following pages are written for all people that try to program in assembler language for the first time and like to start with programming ATMEL-AVRs AT90S, ATtiny, ATmega. Some basic programming techniques are shown that are necessary. During later learning stages these informations are useful and so try to learn this first. Tables are included for easier programming. There might be some bugs in these pages. Bug reports and hints for improving these pages are very welcome.

Overview on the beginners programming tutorial

Theme	What is that	Instructions
Hardware	ISP-Interface, Parallel-Port-Programmer, Experimental 2313 board, Commercial boards	-
Why	Why learn assembler language?, The concept behind assembler, Planning an AVR project	-
Tools	Editor, Assembler, ISP, Studio3, Studio4, Structure	-
Register	What is a Register?	.DEF, LDI, MOV
	Which different registers are available	CLR, ANDI, CBR, CPI, SBCI, SBR, SER, SUBI
	Register as Pointer	LD, ST
	What can be loaded into a register?	-
Ports	What is a Port?	
	Which Ports are available?	-
	Status register as most relevant port	CLx, SEx, BCLR, BSET
SRAM	What is SRAM?	-
	For which purposes SRAM is used	-
	How to use SRAM?	STS, LDS, LD, ST, STD, LDD
	Stack in SRAM	PUSH, POP, RCALL, RET
Program run	What happens during a Reset?	-
	Linear program runs and branches	.CSEG, .DSEG, .ORG, .ESEG, INC, BRNE, BREQ, BRxx
	Timing of commands	NOP, DEC
	Macros	.MACRO, .ENDMACRO
	Subroutines	RET, RCALL, RJMP, SBRC, SBRS, SBIC, SBIS
	Interrupts	RETI
Calculations	Numbers and characters	-
	Bit manipulation	ORI, OR, ANDI, AND, CBR, SBR, EOR, COM, NEG, BLD, CLT, SET, BST
	Shift and rotate	LSL, LSR, ASR, ROL, ROR, SWAP
	Adding, Subtracting, Compares	ADD, ADC, SUB, SBC, CP, CPC, CPI, TST
	Format changes	.DB, .DW
Tables	Commands by function	-
	Commands alphabetic	-
	Ports	-
	Vectors	-
	Abbreviations	-
	Directives	all dot directives
	Expressions	-

Example: What is a Port?

To the top of this page

If you need a complete description of all instructions of the AVRs with explanations you might be happy to have the document here by Anthony Peck. It also provides small projects with an 8-pin ATtiny13.

Source: http://www.avr-asm-tutorial.net/avr_en/beginner/index.html

What is a Port?

Ports in the AVR are gates from the central processing unit to internal and external hard- and software components. The CPU communicates with these components, reads from them or writes to them, e.g. to the timers or the parallel ports. The most used port is the flag register, where results of previous operations are written to and branch conditions are read from.

There are 64 different ports, which are not physically available in all different AVR types. Depending on the storage space and other internal hardware the different ports are either available and accessable or not. Which of these ports can be used is listed in the data sheets for the processor type.

Ports have a fixed address, over which the CPU communicates. The address is independent from the type of AVR. So e.g. the port adress of port B is always 0x18 (0x stands for hexadecimal notation). You don't have to remember these port adresses, they have convenient aliases. These names are defined in the include files (header files) for the different AVR types, that are provided from the producer. The include files have a line defining port B's address as follows:

```
.EQU PORTB, 0x18
```

So we just have to remember the name of port B, not its location in the I/O space of the chip. The include file 8515def.inc is involved by the assembler directive

```
.INCLUDE "C:\Somewhere\8515def.inc"
```

and the registers of the 8515 are all defined then and easily accessable.

Ports usually are organised as 8-bit numbers, but can also hold up to 8 single bits that don't have much to do with each other. If these single bits have a meaning they have their own name associated in the include file, e.g. to enable manipulation of a single bit. Due to that name convention you don't have to remember these bit positions. These names are defined in the data sheets and are given in the include file, too. They are provided here in the port tables.

As an example the MCU General Control Register, called MCUCR, consists of a number of single control bits that control the general property of the chip (see the description in MCUCR in detail). It is a port, fully packed with 8 control bits with their own names (ISC00, ISC01, ...). Those who want to send their AVR to a deep sleep need to know from the data sheet how to set the respective bits. Like this:

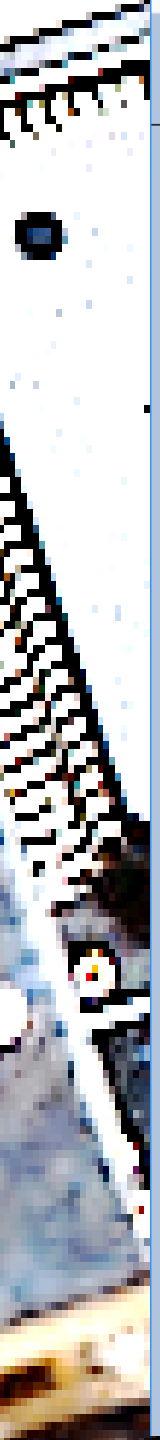
```
.DEF MyPreferredRegister = R16
   LDI MyPreferredRegister, 0b00100000
   OUT MCUCR, MyPreferredRegister
   SLEEP
```

The Out command brings the content of my preferred register, a Sleep-Enable-Bit called SE, to the port MCUCR and sets the AVR immediately to sleep, if there is a SLEEP instruction executed. As all the other bits of MCUCR are also set by the above instructions and the Sleep Mode bit SM was set to zero, a mode called half-sleep will result: no further command execution will be performed but the chip still reacts to timer and other hardware interrupts. These external events interrupt the big sleep of the CPU if they feel they should notify the CPU.

Reading a port's content is in most cases possible using the IN command. The following sequence

```
.DEF MyPreferredRegister = R16
   IN MyPreferredRegister, MCUCR
```

Source: http://www.avr-asm-tutorial.net/avr_en/beginner/index.html



AVR beginners.net

Quick links: Ascii Table | Atmel | AVRFreaks.net

- AVRbeginners.net
  - Getting Started
  - AVR Assembler
  - Converting Numbers (Also asm)
  - AVR Architecture
  - Interfacing Examples
  - Software
  - ATmega8 Dev Board

Created with Adobe GoLive 6.0

updated 4/6/2004
contact webmaster

Welcome to

- Work In Progress -

Crash course on numbers - hex, binary and decimal

-- News --

06-04-2004: LCD 4-bit mode explanation with asm code!

30-03-2004: 24C16 TWI EEPROM interfacing page with C code uploaded.

27-03-2004: LCD pages and our VERY FIRST C Code example!

We also added an introduction to the TWI in the architecture section. Example code is on its way.

26-09-2003: The ATmega8 pages are finished! Here's the intro!

28-08-2003: Added an ATmega8 ADC example!

27-08-2003: The whole site is now available as a zip file!

Source: http://www.avrbeginners.net/

Conditional Branches

Conditonal branches are branches based on the micro's Status Register. If the result of a previous operation left a status (for example "Zero"), this can be used to jump to code handling this result. Loops (for, while...) make use of this.

Any add, subtract, increment, decrement or logic instruction for example leaves a status that can be used for almost any branch instruction the AVR offers. There are as well some tests which set status flags based on their arguments. Basically they are just a subtraction: Comparing two numbers to each other is done by subtracting one from the other. The result of a - b can be negative (b > a), positive (b < a) and zero (b = a). This information is stored in the status register. When two numbers are added to each other, it can happen that the 8-bit result is greater than 255 and therefore "rolls over". In this case, the carry bit in SREG is set.

Some examples:

```
subi r16, 5             ; r16 = r16 - 5
breq r16_is_0           ; r16 was 5, handle that
brlo r16_is_lower0      ; r16 was lower than 5
r16_is_greater5:        ; r16 was higher than 5
```

Now some examples for conditional branches in loops:

```
ldi r16, 5              ; load desired loop count into r16
loop:                   ; loop label
dec r16                 ; decrease loop count
brne loop               ; if not equal (result=0), loop again
                        ;
clr r16                 ; clear counter
loop:                   ; loop label
inc r16                 ; increase loop count
cpi r16, 5              ; compare to desired loop count
brne loop               ; if not reached, loop again
```

Here is a list of simple tests which can also be used for branch instructions (exception: cpse - this instruction performs a compare and skips the next instruction if equal):

				Valid SREG flags after test:							
instruction:	**arg 1:**	**arg 2:**	**"action":**	I	T	H	S	V	N	Z	C
cpi	reg	const	reg - const	-	-	<>	<>	<>	<>	<>	<>
cp	reg	reg	reg - reg	-	-	<>	<>	<>	<>	<>	<>
cpc	reg	reg	reg - reg - C	-	-	<>	<>	<>	<>	<>	<>
tst	reg	---	reg AND reg	-	-	-	<>	0	<>	<>	-
cpse	reg	reg	reg - reg	-	-	-	-	-	-	-	-

Source: http://www.avrbeginners.net/

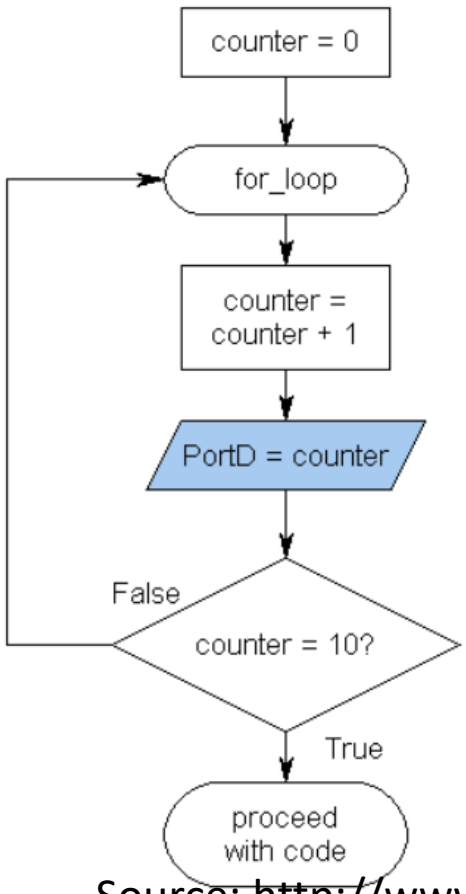
If you want more info on which results change which SREG flag, see the AVRStudio assembler help. Here is a

Flow charts are a graphical representation of code, Program states or even SRAM contents, if used in a creative way. Once you know how to use them for code you'll quickly develop your own style to create flow charts for almost anything.

When you think about implementing a special algorithm or peripheral driver it might be better to have a flow chart already done before you start hacking code. That will save lots of time. Trust me. I know. If you have code that is not sufficiently commented or just BIG, analyse it by making up a flow chart. Very often that helps, especially when you got it from the web.

Especially when writing code in assembler they are a great help, because assembler instructions are not always self-explanatory and even well-structured code will get hard to read once it has grown to a certain size.

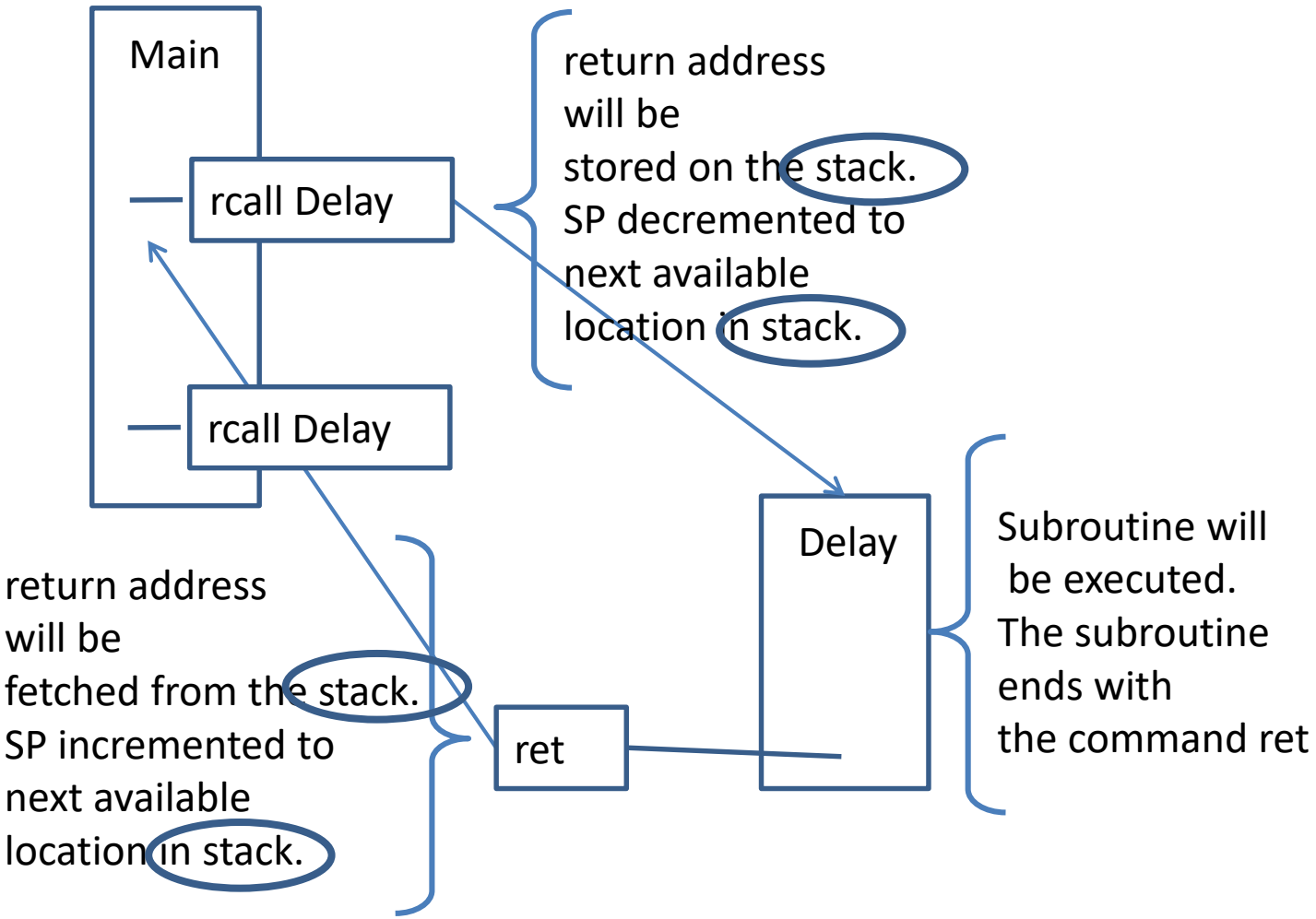
Here is a small example flow chart:



You can find a good program for editing flow charts at www.rff.com. But a piece of paper will do the job too if you need to make up one.

Source: http://www.avrbeginners.net/

Subroutine call, example.



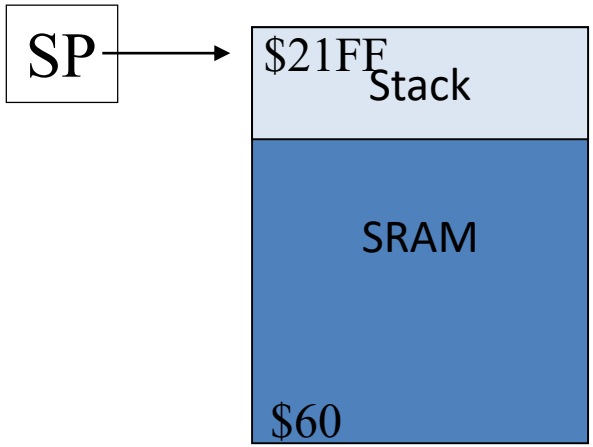
Stack

Stack

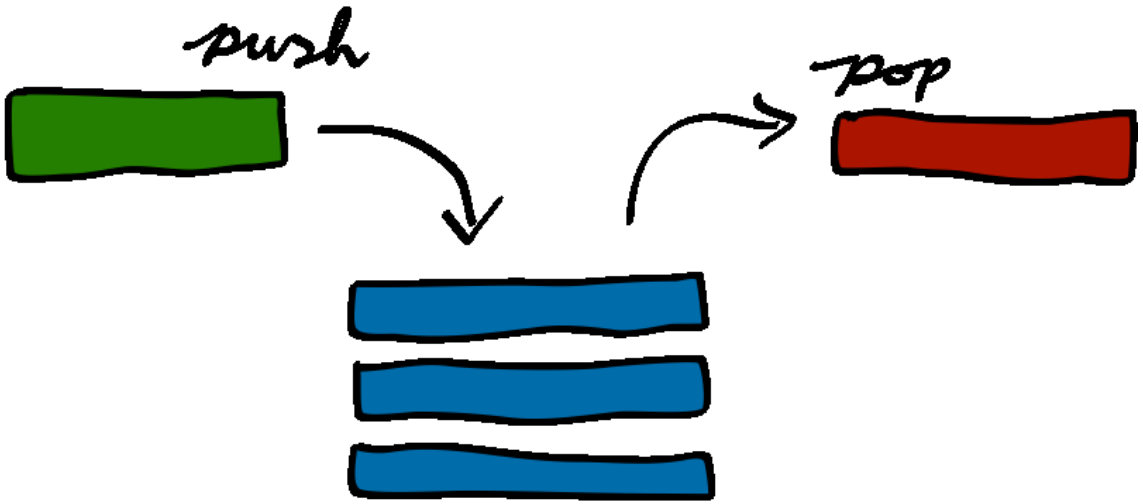
- The stack is an area of SRAM that grows downward from a fixed reference point
  - Special instructions and I/O register support stack access
    - RET, CALL, PUSH, POP, …
    - SP (Stack Pointer)
  - The stack requires initialization
    - Set SP to an appropriate address (empty stack)
  - The stack is managed by application and is used by the processor interrupt system

Initiate the Stack Pointer, SP

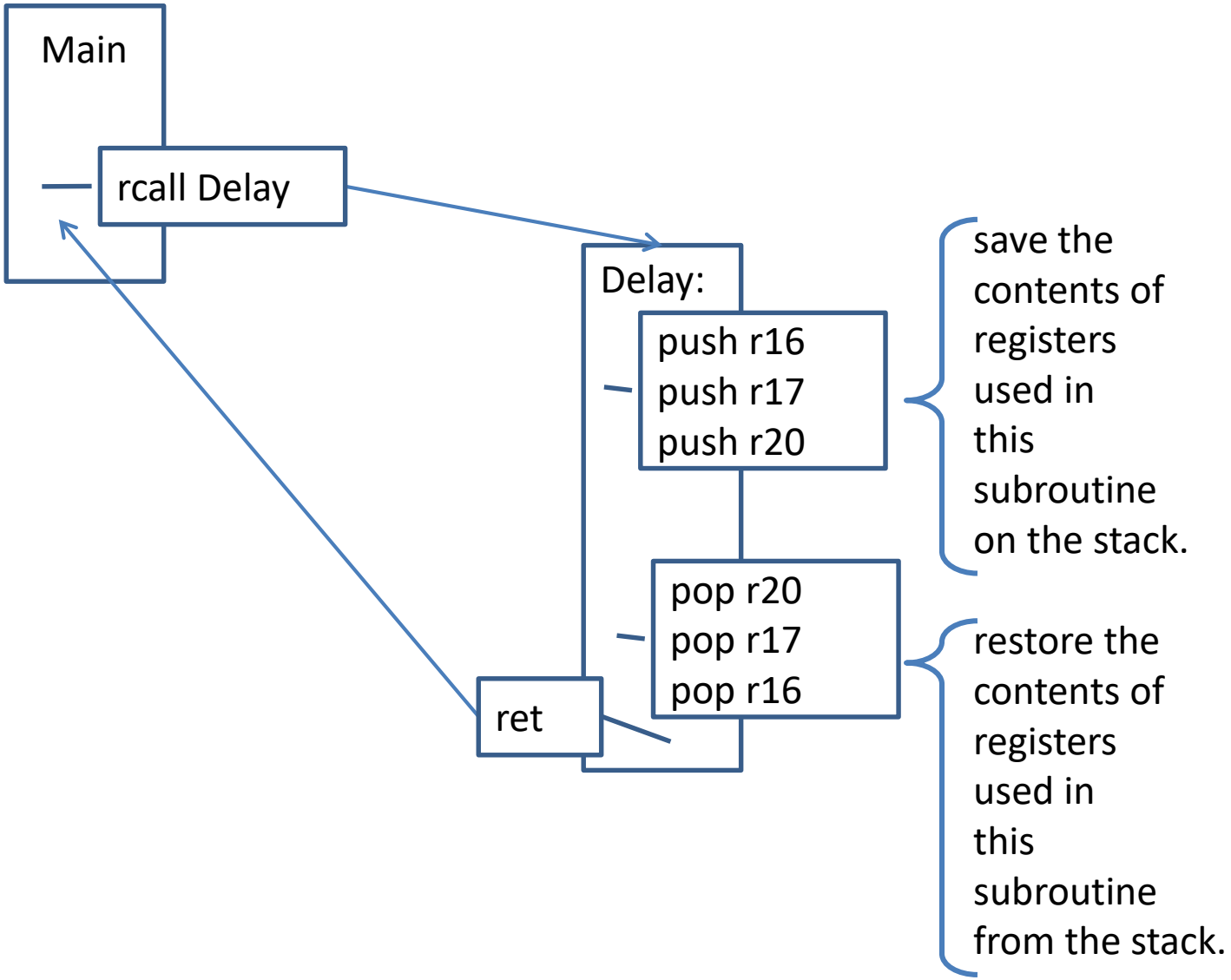
- Typically, the stack starts at the highest SRAM address and grows downward through memory
- SP points to the next available byte of stack storage (top of stack after a push)
- Initialization:
  - ldi R16, high(RAMEND)
  - out SPH, R16
  - ldi R16, low(RAMEND)
  - out SPL, R16
- RAMEND = highest address in RAM



push and pop, for store and restore register contents



Subroutine call, example.



PUSH and POP

- push Rr
- Register data copied to byte at address in SP
- SP is decremented

- pop Rd
- SP is incremented
- Byte at address in SP is copied to Rd

PUSH

Address:	Content:
45A	2C
45B	34
45C	36
45D	11
45E	F2
45F	5C

SP:45D

Top of stack

push R5

Register:	Content:
R3	7F
R4	00
R5	A2
R6	77

POP

Address:	Content:
45A	2C
45B	34
45C	36
45D	A2
45E	F2
45F	5C

SP:45C

Top of stack

pop R4

Register:	Content:
R3	7F
R4	00
R5	A2
R6	77

Stack Usage

- Temporary storage of information
- Storage of return addresses for procedure calls and interrupts
- Storage for local variables during execution of a procedure

Temporary Storage

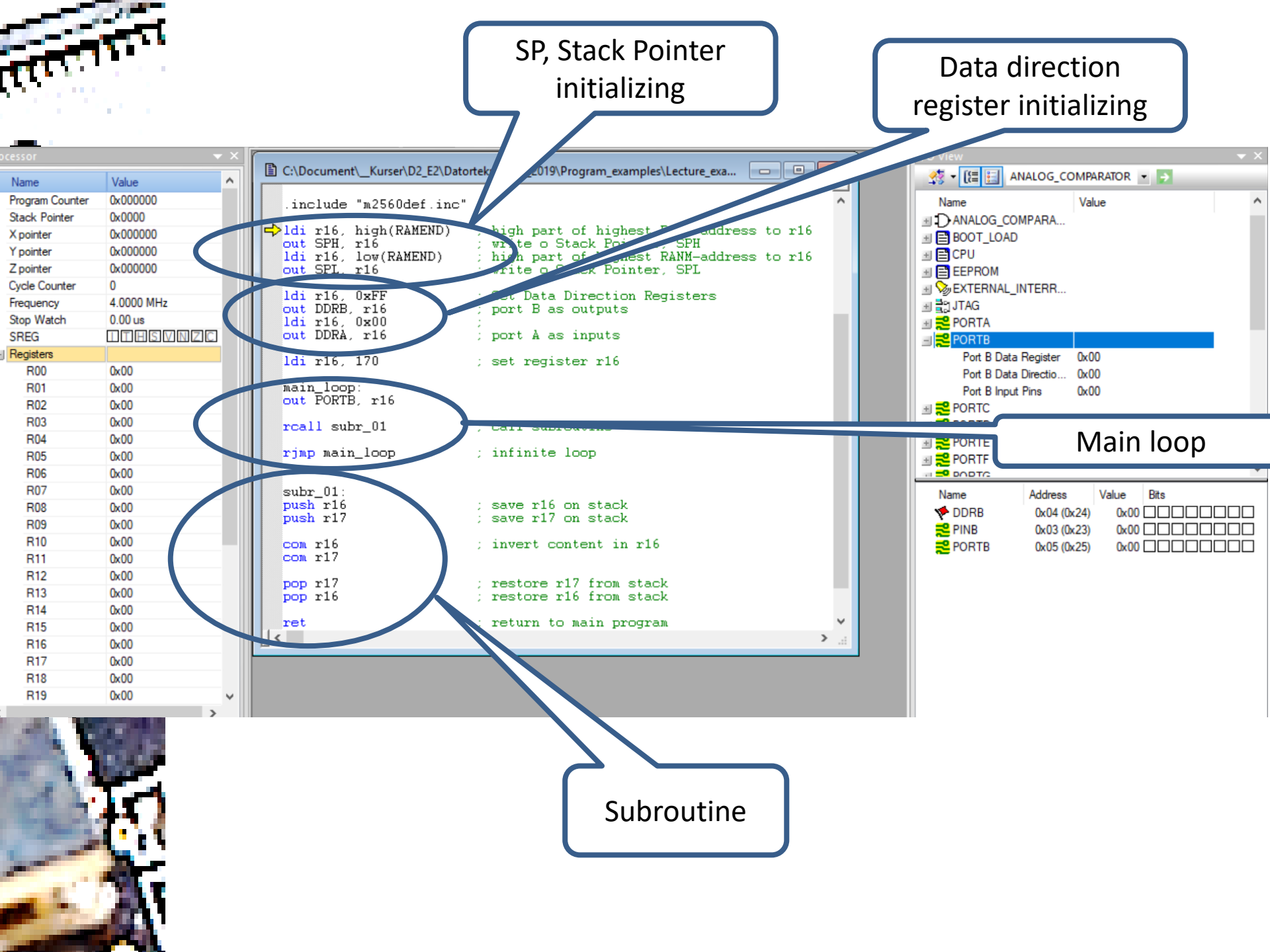
- push R16 ;save register
- ldi R16, $FF ;use register
- out PORTB, R16
- pop R16 ;restore register

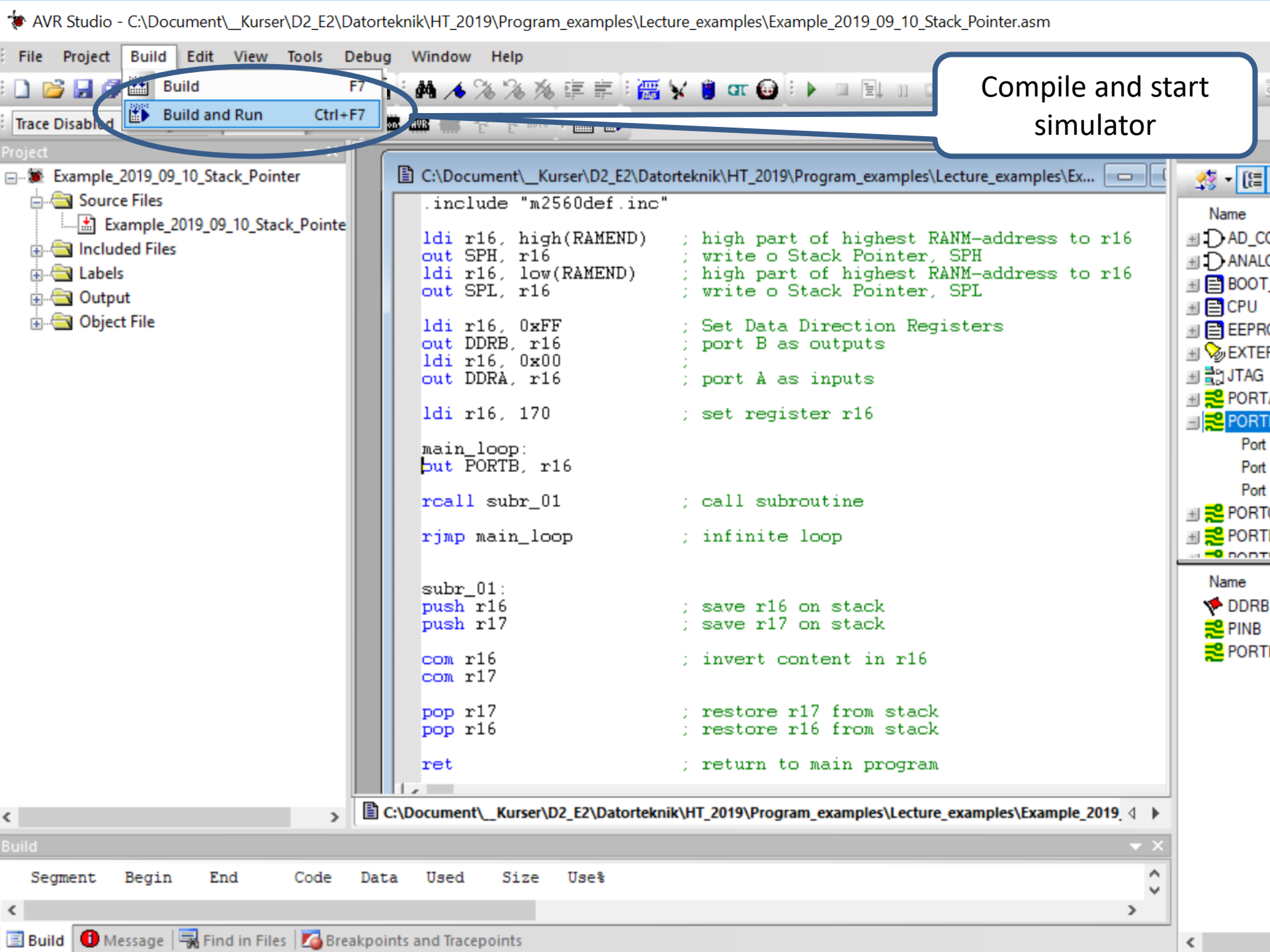
Program example

- Simple program with subroutine call
- Saving register contents on stack in subroutine

C:\Document_Kurser\D2_E2\Datorteknik\HT_2019\Program_examples\Lecture_examples\Ex...

```
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;    1DT301, Computer Technology I
;    Date: 2019-09-09
;    Author:
;        Anders Haggren
;    Function:
;
;        Lecture examples 2019-09-10, lecture #3
;
;        Use of Stack and Stack Pointer setup
;
;
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

.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
ldi r16, 0x00              ;
out DDRA, r16              ; port A as inputs

ldi r16, 170               ; set register r16

main_loop:
out PORTB, r16

rcall subr_01              ; call subroutine

rjmp main_loop             ; infinite loop
```


SP, Stack Pointer initializing
Data direction register initializing
Main loop
Subroutine
Name
Value
Program Counter
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
Registers
R00
R01
R02
R03
R04
R05
R06
R07
R08
R09
R10
R11
R12
R13
R14
R15
R16
R17
R18
R19
0x00
C:\Document_Kurser\D2_E2\Datorteknik_2019\Program_examples\Lecture_exa...
.include "m2560def.inc"
ldi r16, high(RAMEND) ; high part of highest RAM-address to r16
out SPH, r16 ; write o Stack Pointer, SPH
ldi r16, low(RAMEND) ; high part of highest RAM-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
ldi r16, 0x00 ;
out DDRA, r16 ; port A as inputs
ldi r16, 170 ; set register r16
main_loop:
out PORTB, r16
rcall subr_01 ; call subroutine
rjmp main_loop ; infinite loop
subr_01:
push r16 ; save r16 on stack
push r17 ; save r17 on stack
com r16 ; invert content in r16
com r17
pop r17 ; restore r17 from stack
pop r16 ; restore r16 from stack
ret ; return to main program
View
ANALOG_COMPARATOR
Name
Value
ANALOG_COMPARA...
BOOT_LOAD
CPU
EEPROM
EXTERNAL_INTERR...
JTAG
PORTA
PORTB
Port B Data Register
0x00
Port B Data Directio...
0x00
Port B Input Pins
0x00
PORTC
PORTE
PORTF
Name
Address
Value
Bits
DDRB
0x04 (0x24)
0x00
PINB
0x03 (0x23)
0x00
PORTB
0x05 (0x25)
0x00


AVR Studio - C:\Document__Kurser\D2_E2\Datorteknik\HT_2019\Program_examples\Lecture_examples\Example_2019_09_10_Stack_Pointer.asm
File Project Build Edit View Tools Debug Window Help
Build F7
Build and Run Ctrl+F7
Compile and start simulator
Trace Disabled
Project
Example_2019_09_10_Stack_Pointer
Source Files
Example_2019_09_10_Stack_Pointe
Included Files
Labels
Output
Object File
C:\Document__Kurser\D2_E2\Datorteknik\HT_2019\Program_examples\Lecture_examples\Ex...
.include "m2560def.inc"
ldi r16, high(RAMEND) ; high part of highest RAM-address to r16
out SPH, r16 ; write o Stack Pointer, SPH
ldi r16, low(RAMEND) ; high part of highest RAM-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
ldi r16, 0x00 ;
out DDRA, r16 ; port A as inputs
ldi r16, 170 ; set register r16
main_loop:
out PORTB, r16
rcall subr_01 ; call subroutine
rjmp main_loop ; infinite loop
subr_01:
push r16 ; save r16 on stack
push r17 ; save r17 on stack
com r16 ; invert content in r16
com r17
pop r17 ; restore r17 from stack
pop r16 ; restore r16 from stack
ret ; return to main program
C:\Document__Kurser\D2_E2\Datorteknik\HT_2019\Program_examples\Lecture_examples\Example_2019
Name
AD_C
ANALO
BOOT
CPU
EEPR
EXTE
JTAG
PORT
PORT
Port
Port
Port
PORT
PORT
Name
DDRB
PINB
PORT
Build
Segment Begin End Code Data Used Size Use%
Build Message Find in Files Breakpoints and Tracepoints

Name	Value
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	0x00
R17	0x00
R18	0x00
R19	0x00
R20	0x00
R21	0x00
R22	0x00
R23	0x00
R24	0x00
R25	0x00

Processor

C:\Document__Kurser\D2_E2\Datorteknik\HT_2019\Program_examples\Lecture_examples\Ex...

```
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;   1DT301, Computer Technology I
;   Date: 2019-09-09
;   Author:
;       Anders Haggren
;   Function:
;
;       Lecture examples 2019-09-10, lecture #3
;
;       Use of Stack and Stack Pointer setup
;
;
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

 include "m2560def.inc"

ldi r16, high(RAMEND)     ; high p
out SPH, r16              ; write o Stack Pointer, SPH
ldi r16, low(RAMEND)      ; high part of highest RAM-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
ldi r16, 0x00             ;
out DDRA, r16             ; port A as inputs

ldi r16, 170              ; set register r16

main_loop:
out PORTB, r16

rcall subr_01             ; call subroutine

rjmp main_loop            ; infinite loop
```

C:\Document__Kurser\D2_E2\Datorteknik\HT_2019\Program_examples\Lecture_examples\Example_2019_09_10_Stacl

I/O View

ANALOG_COMPARATOR

Name	Value
AD_CONVERTER	
ANALOG_COMPARA...	
BOOT_LOAD	
CPU	
EEPROM	
EXTERNAL_INTERR...	
JTAG	

Name	Address	Value	Bits
DDRB	0x04 (0x24)	0x00	
PINB	0x03 (0x23)	0x00	
PORTB	0x05 (0x25)	0x00	

Yellow arrow indicates PC, Program Counter

PC, Program Counter also here

Yellow arrow indicates PC, Pro Counter

Processor

Name	Value
Program Counter	0x000000
Stack Pointer	0x21FF
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
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00

Project | Processor

C:\Document__Kurser\D2_E2\Datorteknik\H

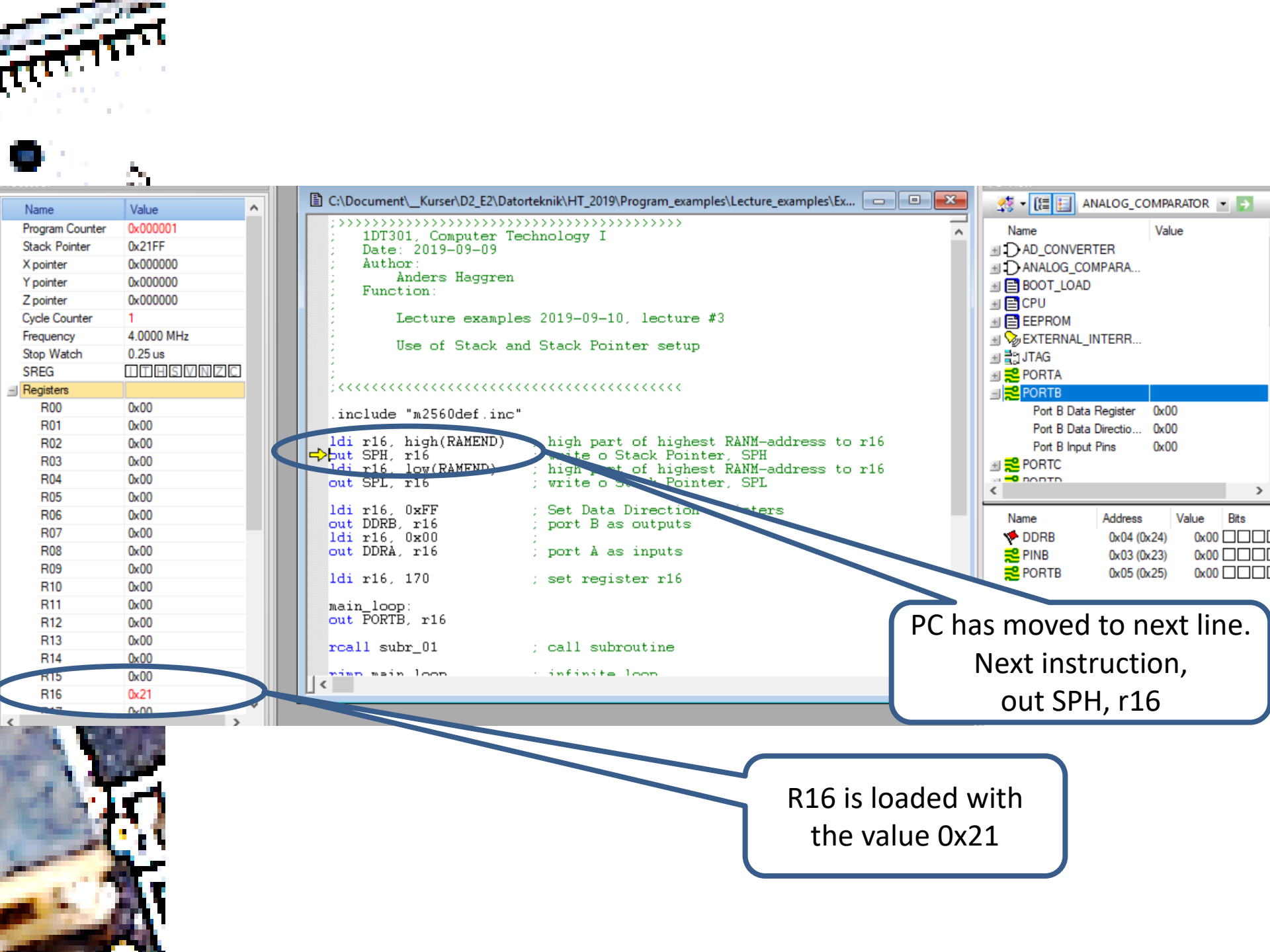
```
;    Date: 2019-09-09
;    Author:
;        Anders Haggren
;    Function:
;
;        Lecture examples 2019-09-10, lecture #3
;
;        Use of Stack and Stack Pointer setup
;
;
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

.include "m2560def.inc"

ldi r16, high(RAMEND)     ; high p
out SPH, r16              ; write o Stack Pointer, SPH
ldi r16, low(RAMEND)      ; high part of highest RAM-address to r1
out SPL, r16              ; write o Stack Pointer, SPL

ldi r16, 0xFF             ; Set Data Direction Registers
out DDRB, r16             ; port B as outputs
ldi r16, 0x00             ;
out DDRA, r16             ; port A as inputs

ldi r16, 170              ; set register r16

main_loop:
out PORTB, r16

rcall subr_01             ; call subroutine

rjmp main_loop            ; infinite loop
```

C:\Document__Kurser\D2_E2\Datorteknik\HT_2019\Program_examples\Lecture_examples\Example_2019_09_10_Stacl

I/O View

Name
AD_CONV
ANALOG_
BOOT_LO
CPU
EEPROM
EXTERNA
JTAG

Name
DDRB
PINB
PORTB


Name Value
Program Counter 0x000001
Stack Pointer 0x21FF
X pointer 0x000000
Y pointer 0x000000
Z pointer 0x000000
Cycle Counter 1
Frequency 4.0000 MHz
Stop Watch 0.25 us
SREG
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 0x21
C:\Document__Kurser\D2_E2\Datorteknik\HT_2019\Program_examples\Lecture_examples\Ex...
;>>
; 1DT301, Computer Technology I
; Date: 2019-09-09
; Author:
; Anders Haggren
; Function:
;
; Lecture examples 2019-09-10, lecture #3
;
; Use of Stack and Stack Pointer setup
;
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
.include "m2560def.inc"
ldi r16, high(RAMEND) ; high part of highest RAM-address to r16
out SPH, r16 ; write o Stack Pointer, SPH
ldi r16, low(RAMEND) ; high part of highest RAM-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
ldi r16, 0x00 ;
out DDRA, r16 ; port A as inputs
ldi r16, 170 ; set register r16
main_loop:
out PORTB, r16
rcall subr_01 ; call subroutine
rjmp main_loop ; infinite loop
ANALOG_COMPARATOR
Name Value
AD_CONVERTER
ANALOG_COMPARA...
BOOT_LOAD
CPU
EEPROM
EXTERNAL_INTERR...
JTAG
PORTA
PORTB
Port B Data Register 0x00
Port B Data Directio... 0x00
Port B Input Pins 0x00
PORTC
Name Address Value Bits
DDRB 0x04 (0x24) 0x00
PINB 0x03 (0x23) 0x00
PORTB 0x05 (0x25) 0x00
PC has moved to next line.
Next instruction,
out SPH, r16
R16 is loaded with
the value 0x21

Processor

Name	Value
Program Counter	0x000001
Stack Pointer	0x0000
X pointer	0x000000
Y pointer	0x000000
Z pointer	0x000000
Cycle Counter	1
Frequency	4.0000 MHz
Stop Watch	0.25 us
SREG	[ ][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	0x21
R17	0x00

C:\Document__Kurser\D2_E2\Datorteknik\HT_2019\Program_examples\Lecture_exa...

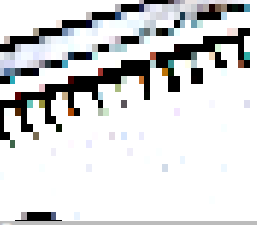
```
ldi r16, high(RAMEND)    ; high part of highest RAM-address to r16
out SPH, r16             ; write o Stack Pointer, SPH
ldi r16, low(RAMEND)     ; high part of highest RAM-address to r16
out SPL, r16             ; write o Stack Pointer, SPL

ldi r16, 0xFF            ; Set Data Direction Registers
out DDRB, r16            ; port B as outputs
ldi r16, 0x00            ;
out DDRA, r16            ; port A as inputs

ldi r16, 170             ; set register r16

main_loop:
out PORTB, r16

rcall subr_01            ; call subroutine

rjmp main_loop           ; infinite loop


subr_01:
push r16                 ; save r16 on stack
push r17                 ; save r17 on stack

com r16                  ; invert content in r16
com r17

pop r17                  ; restore r17 from stack
pop r16                  ; restore r16 from stack

ret                      ; return to main program
```

I/O View

ANALOG_COMPARA

Name	Value
AD_CONVERTER	
ANALOG_COMPARA...	
BOOT_LOAD	
CPU	
EEPROM	
EXTERNAL_INTERR...	
JTAG	
PORTA	
PORTB	
PORTC	
PORTD	
PORTE	
PORTF	
PORTG	

Name	Address	Va



Processor

Name	Value
Program Counter	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	0x21
R17	0x00

C:\Document__Kurser\D2_E2\Datorteknik\HT_2019\Program_examples\Lecture_exa...

```
ldi r16, high(RAMEND)    ; high part of highest RAM-address to r16
out SPH, r16             ; write o Stack Pointer, SPH
ldi r16, low(RAMEND)     ; high part of highest RAM-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
ldi r16, 0x00            ;
out DDRA, r16            ; port A as inputs

ldi r16, 170             ; set register r16

main_loop:
out PORTB, r16

rcall subr_01            ; call subroutine

rjmp main_loop           ; infinite loop


subr_01:
push r16                 ; save r16 on stack
push r17                 ; save r17 on stack

com r16                  ; invert content in r16
com r17

pop r17                  ; restore r17 from stack
pop r16                  ; restore r16 from stack

ret                      ; return to main program
```

I/O View

ANALOG_CO

Name

- AD_CONVERTER
- ANALOG_COMPARA...
- BOOT_LOAD
- CPU
- EEPROM
- EXTERNAL_INTERR...
- JTAG
- PORTA
- PORTB
- PORTC
- PORTD
- PORTE
- PORTF
- PORTG

Name | Address

Processor

Name	Value
Program Counter	0x000003
Stack Pointer	0x2100
X pointer	0x000000
Y pointer	0x000000
Z pointer	0x000000
Cycle Counter	3
Frequency	4.0000 MHz
Stop Watch	0.75 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
R17	0x00

C:\Document__Kurser\D2_E2\Datorteknik\HT_2019\Program_examples\Lecture_exa...

```
ldi r16, high(RAMEND)    ; high part of highest RAM-address to r16
out SPH, r16             ; write o Stack Pointer, SPH
ldi r16, low(RAMEND)     ; high part of highest RAM-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
ldi r16, 0x00            ;
out DDRA, r16            ; port A as inputs

ldi r16, 170             ; set register r16

main_loop:
out PORTB, r16

rcall subr_01            ; call subroutine

rjmp main_loop           ; infinite loop


subr_01:
push r16                 ; save r16 on stack
push r17                 ; save r17 on stack

com r16                  ; invert content in r16
com r17

pop r17                  ; restore r17 from stack
pop r16                  ; restore r16 from stack

ret                      ; return to main program
```

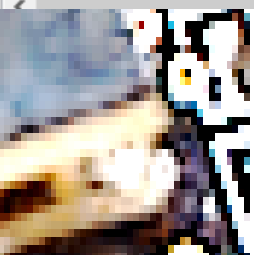
I/O View

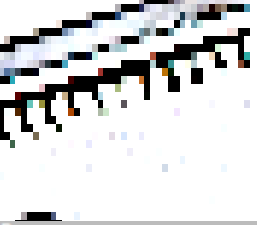
ANALOG_C

Name

- AD_CONVERTER
- ANALOG_COMPARA...
- BOOT_LOAD
- CPU
- EEPROM
- EXTERNAL_INTERR...
- JTAG
- PORTA
- PORTB
- PORTC
- PORTD
- PORTE
- PORTF
- PORTG

Name | Address





Processor

Name	Value
Program Counter	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
R17	0x00

C:\Document__Kurser\D2_E2\Datorteknik\HT_2019\Program_examples\Lecture_exa...

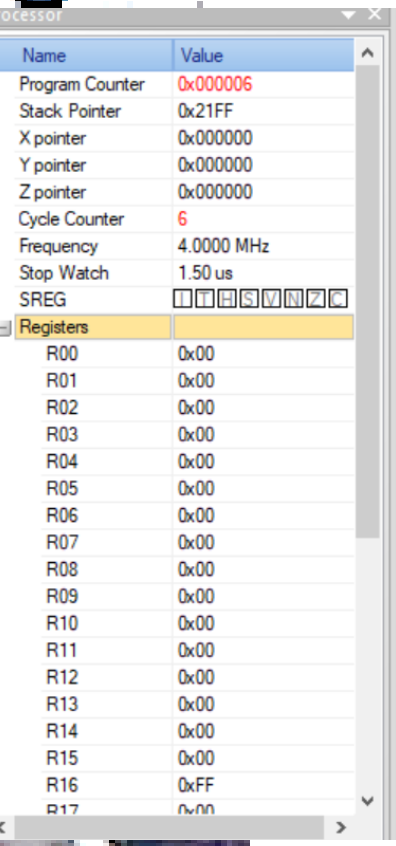
```
ldi r16, high(RAMEND)    ; high part of highest RAM-address to r16
out SPH, r16             ; write o Stack Pointer, SPH
ldi r16, low(RAMEND)     ; high part of highest RAM-address to r16
out SPL, r16             ; write o Stack Pointer, SPL

ldi r16, 0xFF            ; Set Data Direction Registers
out DDRB, r16            ; port B as outputs
ldi r16, 0x00            ;
out DDRA, r16            ; port A as inputs

ldi r16, 170             ; set register r16

main_loop:
out PORTB, r16

rcall subr_01            ; call subroutine

rjmp main_loop           ; infinite loop


subr_01:
push r16                 ; save r16 on stack
push r17                 ; save r17 on stack

com r16                  ; invert content in r16
com r17

pop r17                  ; restore r17 from stack
pop r16                  ; restore r16 from stack

ret                      ; return to main program
```

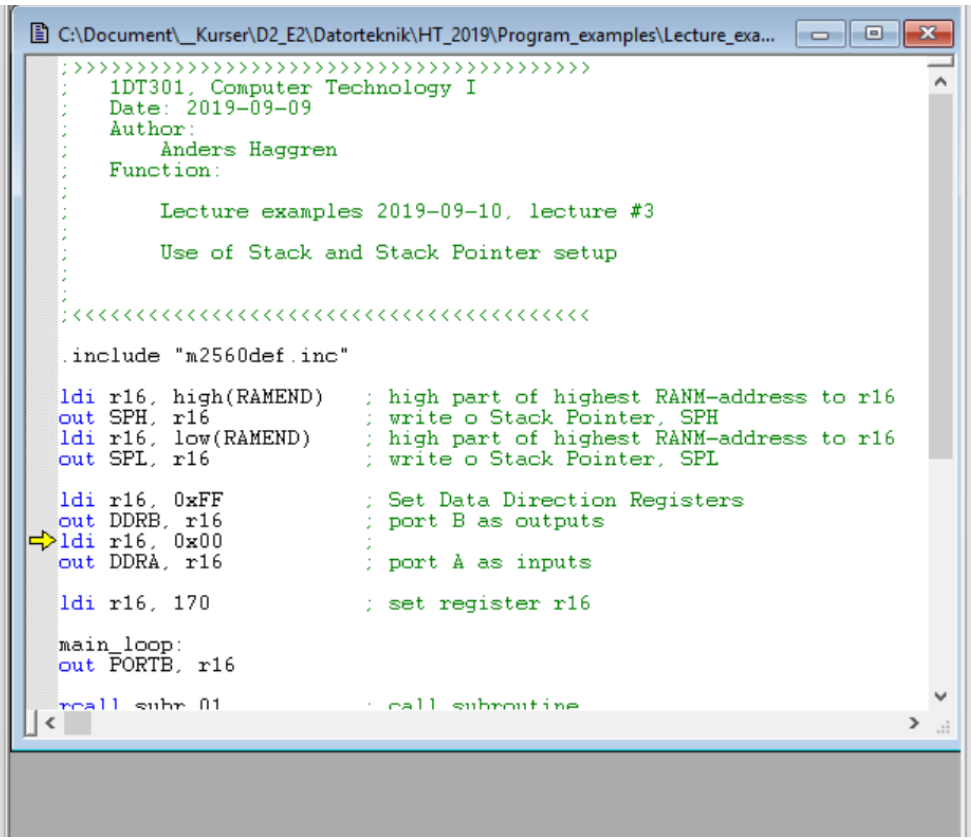
I/O View

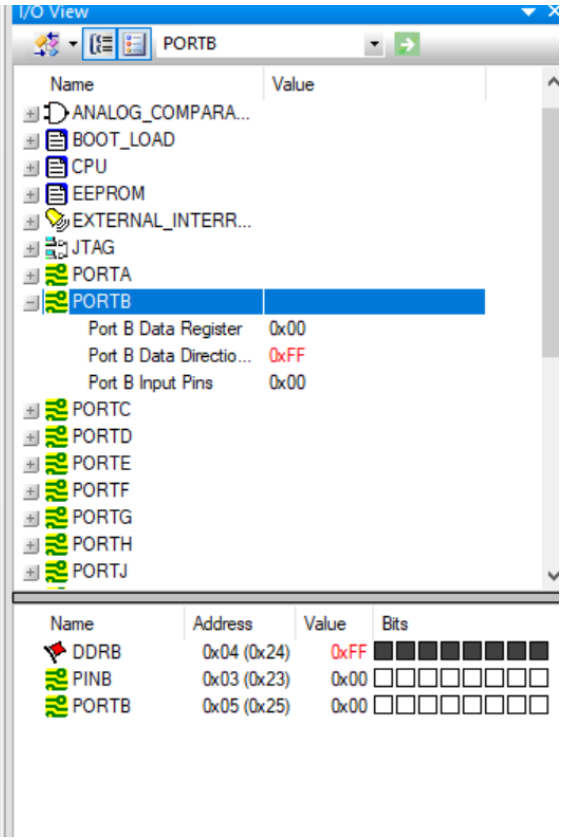
ANALOG_COMPARA

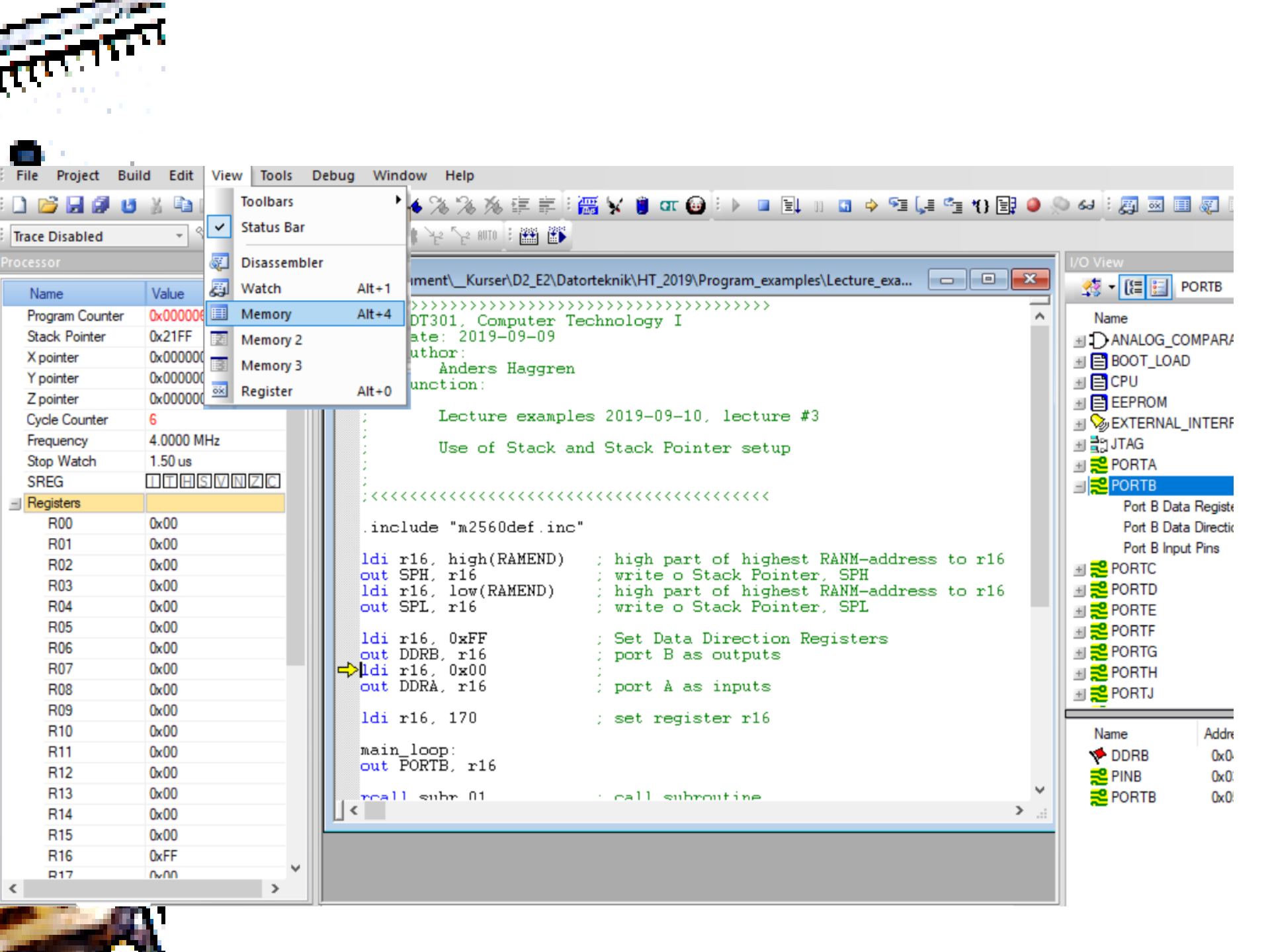
Name	Value
AD_CONVERTER	
ANALOG_COMPARA...	
BOOT_LOAD	
CPU	
EEPROM	
EXTERNAL_INTERR...	
JTAG	
PORTA	
PORTB	
PORTC	
PORTD	
PORTE	
PORTF	
PORTG	

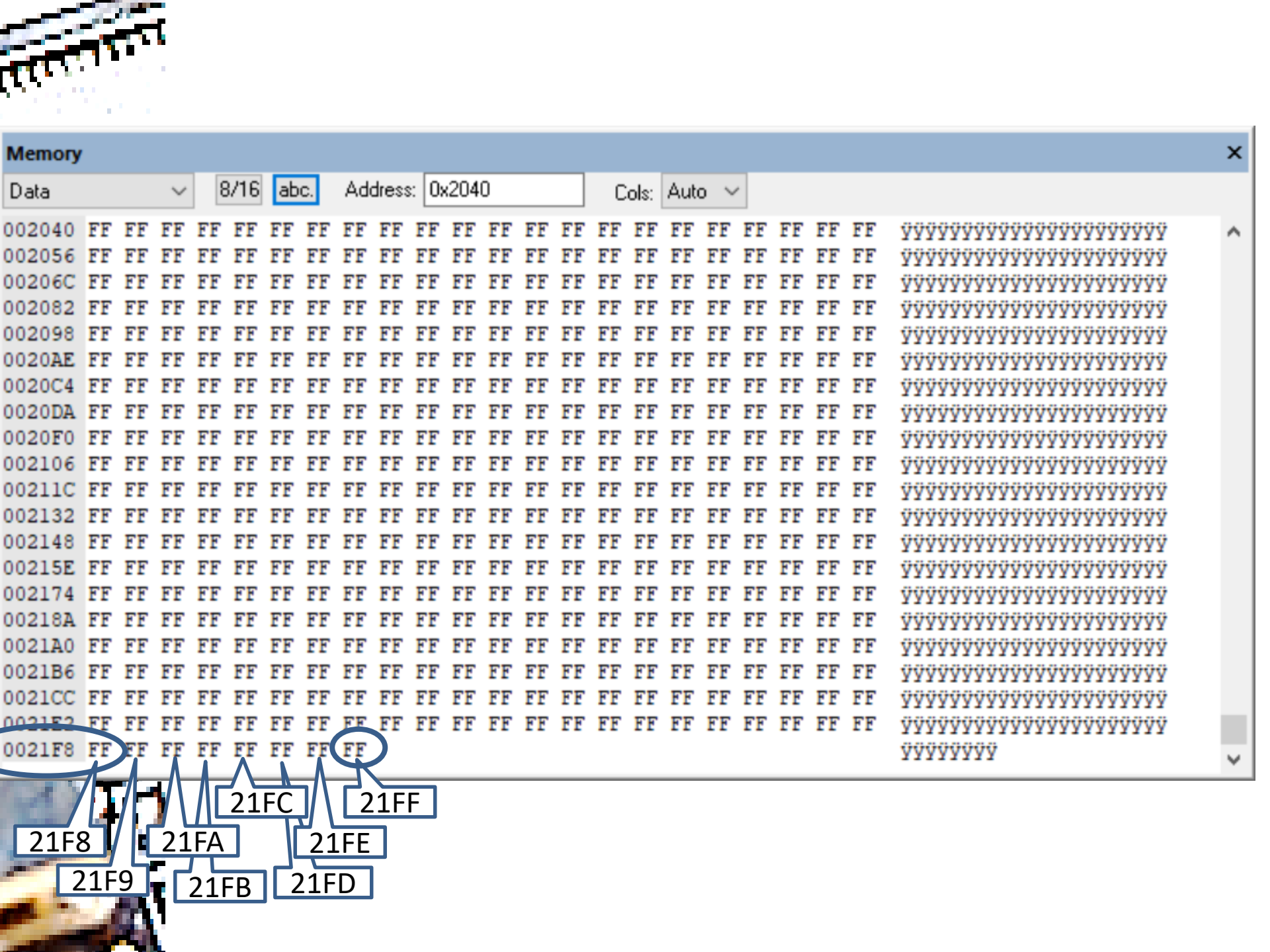
Name	Address	Va

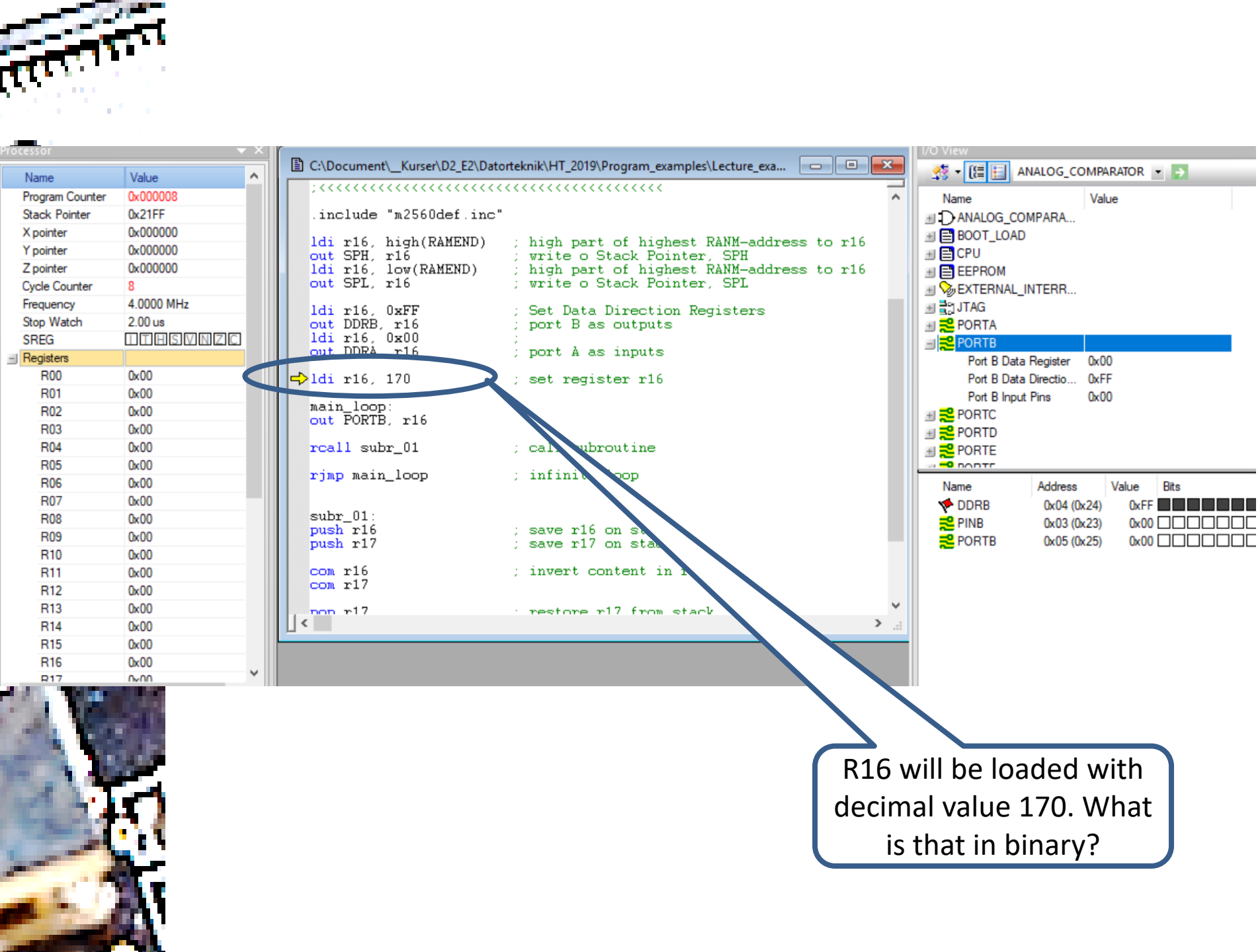

Name
Value
Program Counter
0x000006
Stack Pointer
0x21FF
X pointer
0x000000
Y pointer
0x000000
Z pointer
0x000000
Cycle Counter
6
Frequency
4.0000 MHz
Stop Watch
1.50 us
SREG
Registers
R00
0x00
R01
0x00
R02
0x00
R03
0x00
R04
0x00
R05
0x00
R06
0x00
R07
0x00
R08
0x00
R09
0x00
R10
0x00
R11
0x00
R12
0x00
R13
0x00
R14
0x00
R15
0x00
R16
0xFF


C:\Document__Kurser\D2_E2\Datorteknik\HT_2019\Program_examples\Lecture_exa...
;>>
; 1DT301, Computer Technology I
; Date: 2019-09-09
; Author:
; Anders Haggren
; Function:
;
; Lecture examples 2019-09-10, lecture #3
;
; Use of Stack and Stack Pointer setup
;
;
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
.include "m2560def.inc"
ldi r16, high(RAMEND) ; high part of highest RAM-address to r16
out SPH, r16 ; write o Stack Pointer, SPH
ldi r16, low(RAMEND) ; high part of highest RAM-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
ldi r16, 0x00 ;
out DDRA, r16 ; port A as inputs
ldi r16, 170 ; set register r16
main_loop:
out PORTB, r16
rcall subr_01 ; call subroutine

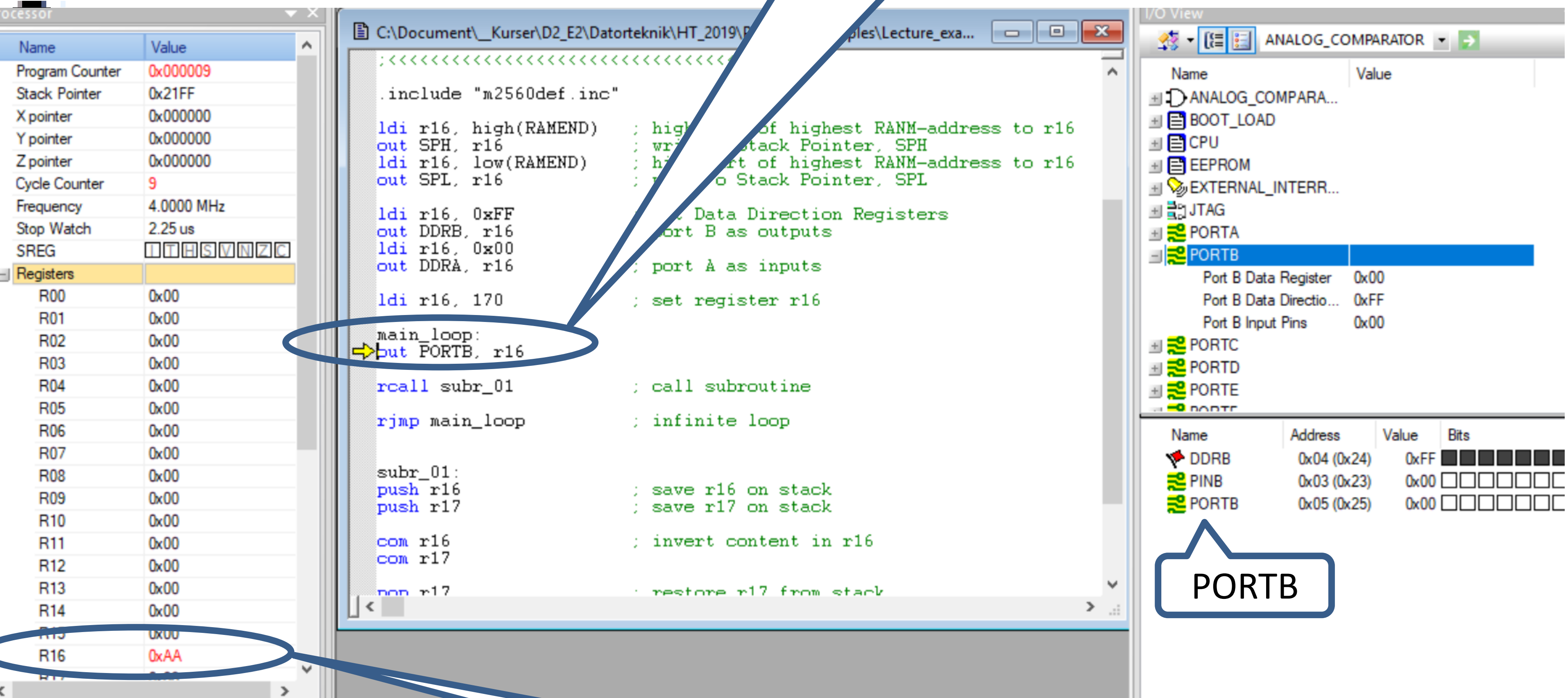

I/O View
PORTB
Name
Value
ANALOG_COMPARA...
BOOT_LOAD
CPU
EEPROM
EXTERNAL_INTERR...
JTAG
PORTA
PORTB
Port B Data Register
0x00
Port B Data Directio...
0xFF
Port B Input Pins
0x00
PORTC
PORTD
PORTE
PORTF
PORTG
PORTH
PORTJ
Name
Address
Value
Bits
DDRB
0x04 (0x24)
0xFF
PINB
0x03 (0x23)
0x00
PORTB
0x05 (0x25)
0x00


File Project Build Edit View Tools Debug Window Help
Toolbars
Status Bar
Disassembler
Watch Alt+1
Memory Alt+4
Memory 2
Memory 3
Register Alt+0
Trace Disabled
Processor
Name Value
Program Counter
Stack Pointer 0x21FF
X pointer
Y pointer
Z pointer
Cycle Counter 6
Frequency 4.0000 MHz
Stop Watch 1.50 us
SREG
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
DT301, Computer Technology I
Anders Haggren
Lecture examples 2019-09-10, lecture #3
Use of Stack and Stack Pointer setup
.include "m2560def.inc"
ldi r16, high(RAMEND) ; high part of highest RAM-address to r16
out SPH, r16 ; write o Stack Pointer, SPH
ldi r16, low(RAMEND) ; high part of highest RAM-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
ldi r16, 0x00 ;
out DDRA, r16 ; port A as inputs
ldi r16, 170 ; set register r16
main_loop:
out PORTB, r16
I/O View
PORTB
Name
ANALOG_COMPARA
BOOT_LOAD
CPU
EEPROM
JTAG
PORTA
PORTB
Port B Data Regist
Port B Data Directi
Port B Input Pins
PORTC
PORTD
PORTE
PORTF
PORTG
PORTH
PORTJ
Name
DDRB
PINB
PORTB


Memory
Data
8/16
abc.
Address: 0x2040
Cols: Auto
002040 FF FF FF FF FF FF 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 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 FF FF
002082 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
002098 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0020AE FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0020C4 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0020DA FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0020F0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
002106 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00211C FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
002132 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
002148 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00215E FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
002174 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00218A FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0021A0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0021B6 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0021CC FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0021F8 FF FF FF FF FF FF FF FF
21F8
21F9
21FA
21FB
21FC
21FD
21FE
21FF

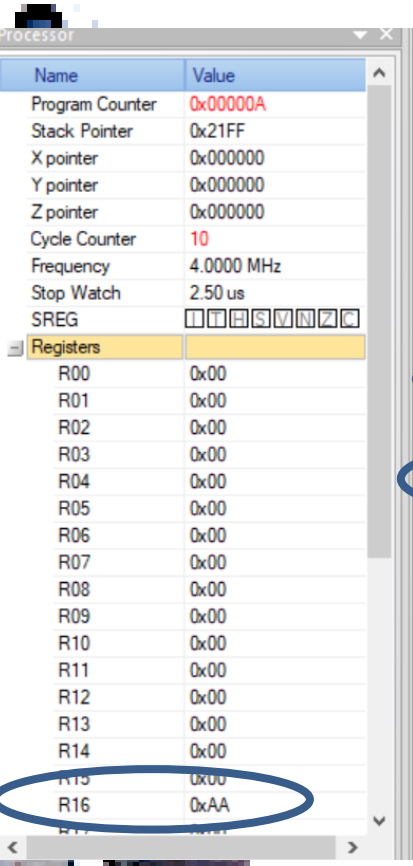

Processor
Name
Value
Program Counter
0x000008
Stack Pointer
0x21FF
X pointer
0x000000
Y pointer
0x000000
Z pointer
0x000000
Cycle Counter
8
Frequency
4.0000 MHz
Stop Watch
2.00 us
SREG
Registers
R00
R01
R02
R03
R04
R05
R06
R07
R08
R09
R10
R11
R12
R13
R14
R15
R16
C:\Document__Kurser\D2_E2\Datorteknik\HT_2019\Program_examples\Lecture_exa...
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
.include "m2560def.inc"
ldi r16, high(RAMEND) ; high part of highest RAM-address to r16
out SPH, r16 ; write o Stack Pointer, SPH
ldi r16, low(RAMEND) ; high part of highest RAM-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
ldi r16, 0x00 ;
out DDRA, r16 ; port A as inputs
ldi r16, 170 ; set register r16
main_loop:
out PORTB, r16
rcall subr_01 ; call subroutine
rjmp main_loop ; infinit loop
subr_01:
push r16 ; save r16 on st
push r17 ; save r17 on stac
com r16 ; invert content in r
com r17
pop r17 ; restore r17 from stack
I/O View
ANALOG_COMPARATOR
Name
Value
ANALOG_COMPARA...
BOOT_LOAD
CPU
EEPROM
EXTERNAL_INTERR...
JTAG
PORTA
PORTB
Port B Data Register
0x00
Port B Data Directio...
0xFF
Port B Input Pins
0x00
PORTC
PORTD
PORTE
Name
Address
Value
Bits
DDRB
0x04 (0x24)
0xFF
PINB
0x03 (0x23)
0x00
PORTB
0x05 (0x25)
0x00
R16 will be loaded with decimal value 170. What is that in binary?

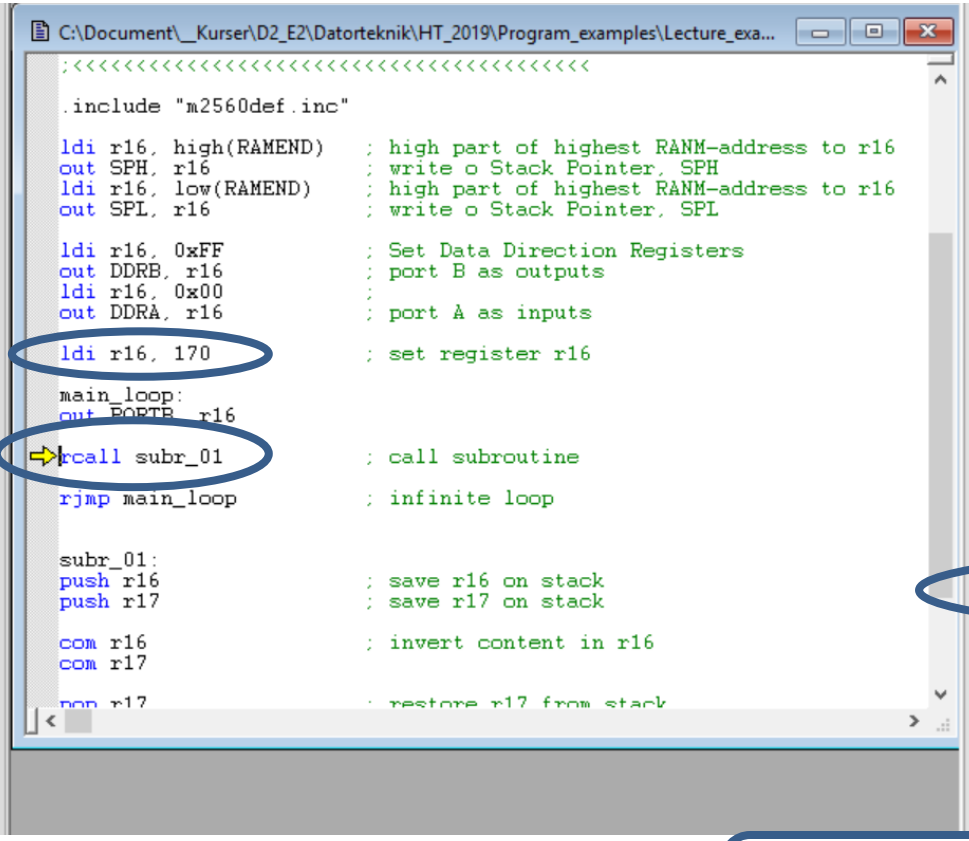
Content of R16 will be copied to PORTB

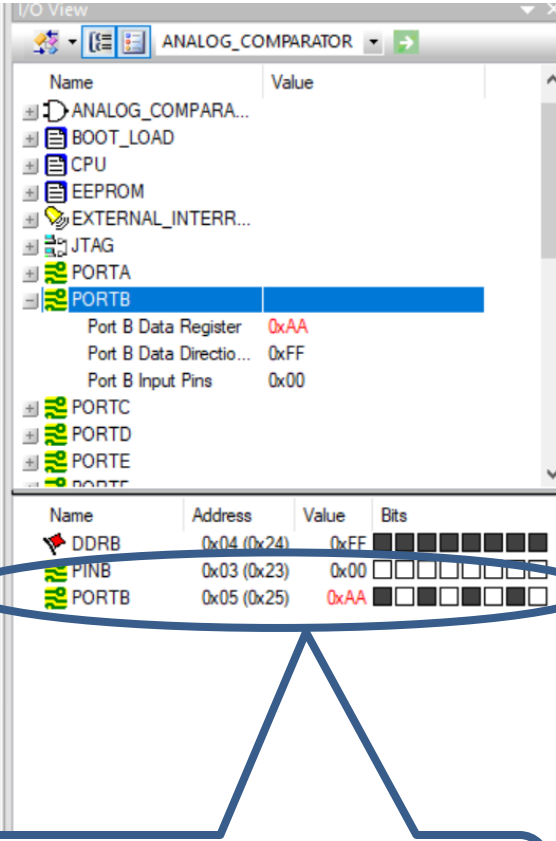


PORTB

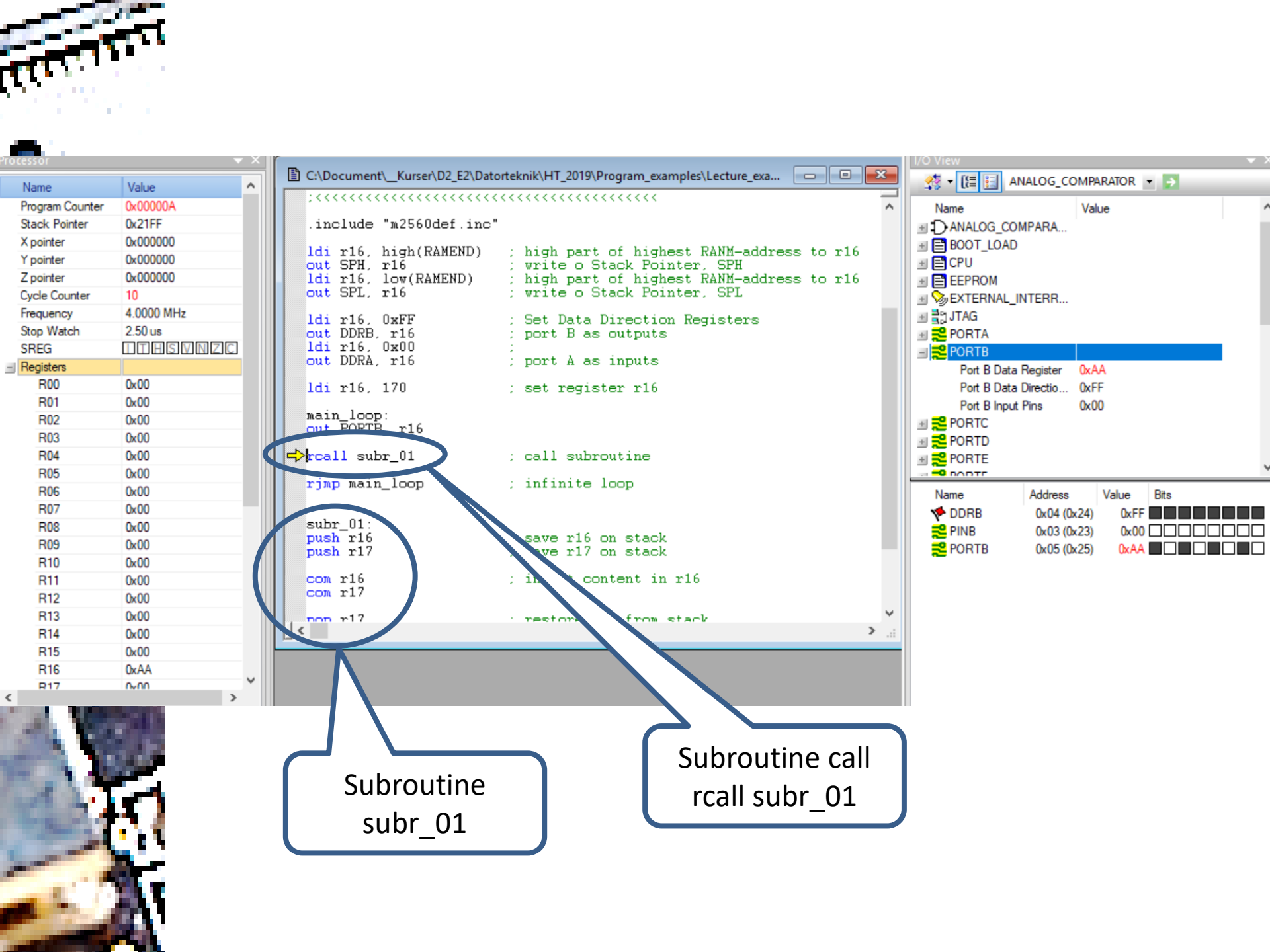
R16 has got hexadecimal value AA which is equal to decimal 170.



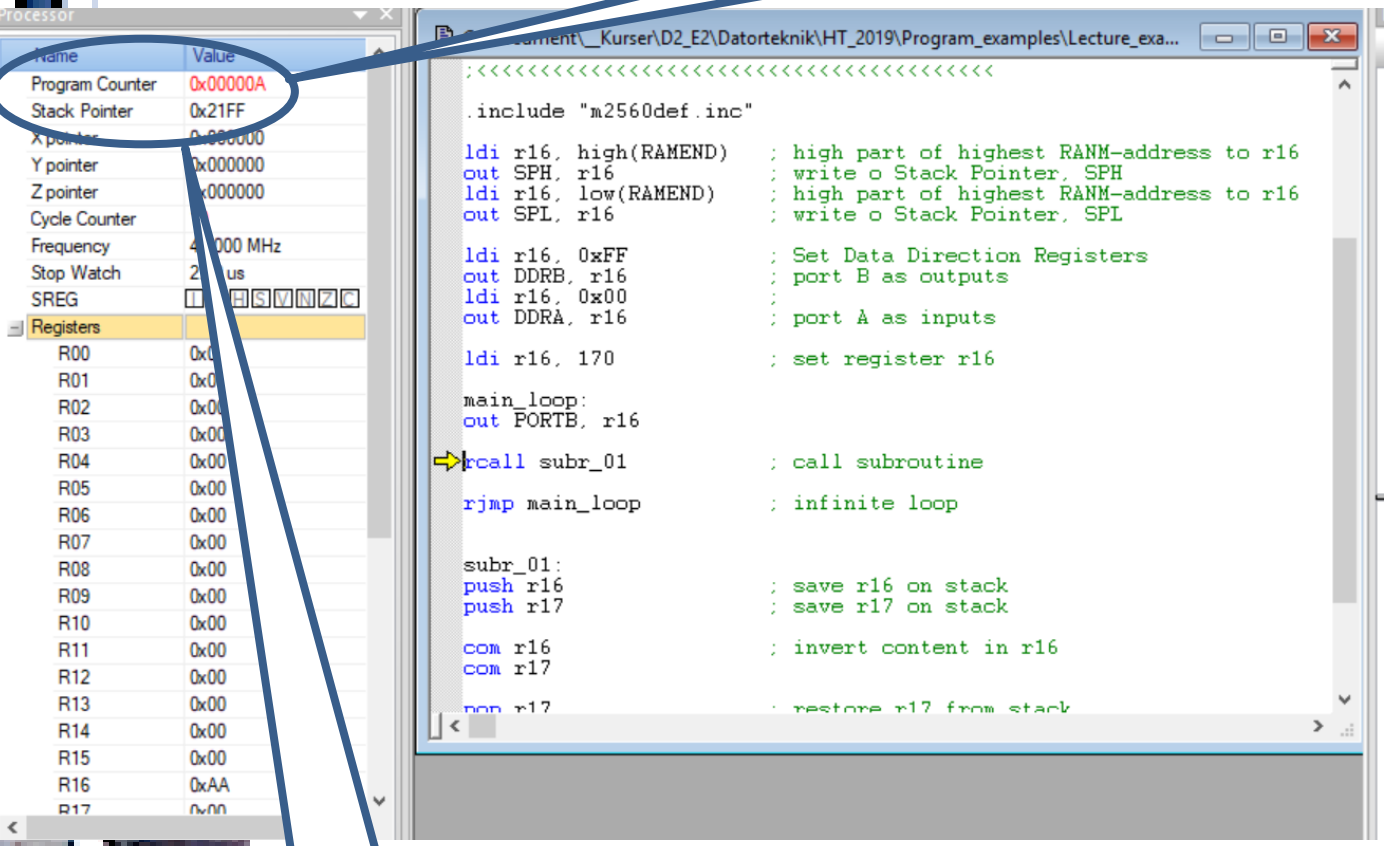




PORTB got the hexadecimal value AA which is binary 1010 1010

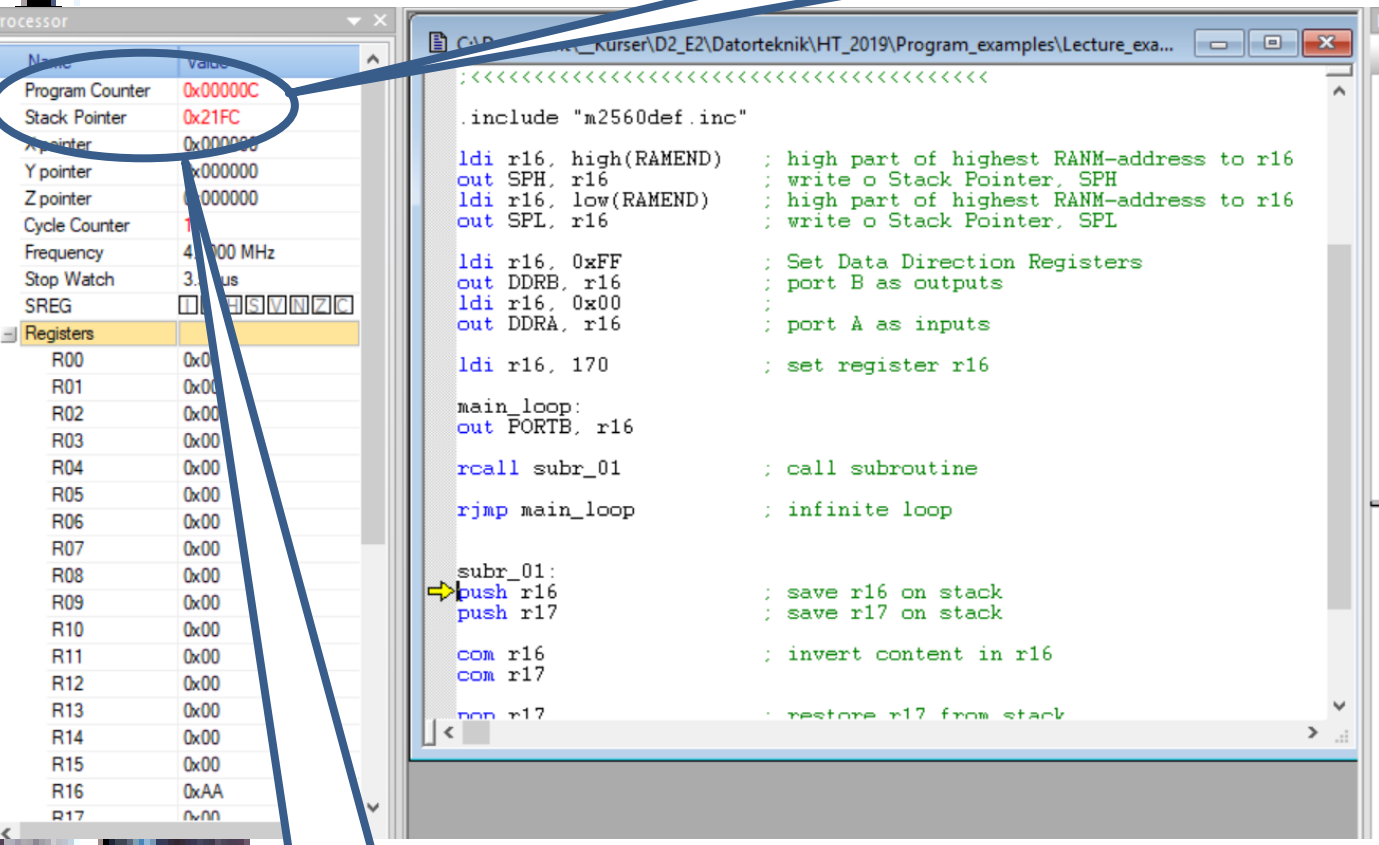

Processor
Name
Value
Program Counter
0x00000A
Stack Pointer
0x21FF
X pointer
0x000000
Y pointer
0x000000
Z pointer
0x000000
Cycle Counter
10
Frequency
4.0000 MHz
Stop Watch
2.50 us
SREG
Registers
R00
R01
R02
R03
R04
R05
R06
R07
R08
R09
R10
R11
R12
R13
R14
R15
R16
0xAA
C:\Document_Kurser\D2_E2\Datorteknik\HT_2019\Program_examples\Lecture_exa...
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
.include "m2560def.inc"
ldi r16, high(RAMEND) ; high part of highest RAM-address to r16
out SPH, r16 ; write o Stack Pointer, SPH
ldi r16, low(RAMEND) ; high part of highest RAM-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
ldi r16, 0x00 ;
out DDRA, r16 ; port A as inputs
ldi r16, 170 ; set register r16
main_loop:
out PORTB, r16
rcall subr_01 ; call subroutine
rjmp main_loop ; infinite loop
subr_01:
push r16 ; save r16 on stack
push r17 ; save r17 on stack
com r16 ; in content in r16
com r17
pop r17 ; restor from stack
I/O View
ANALOG_COMPARATOR
Name
Value
ANALOG_COMPARA...
BOOT_LOAD
CPU
EEPROM
EXTERNAL_INTERR...
JTAG
PORTA
PORTB
Port B Data Register
0xAA
Port B Data Directio...
0xFF
Port B Input Pins
0x00
PORTC
PORTD
PORTE
Name
Address
Value
Bits
DDRB
0x04 (0x24)
0xFF
PINB
0x03 (0x23)
0x00
PORTB
0x05 (0x25)
0xAA
Subroutine
subr_01
Subroutine call
rcall subr_01

PC, Program Counter = 0x00000A

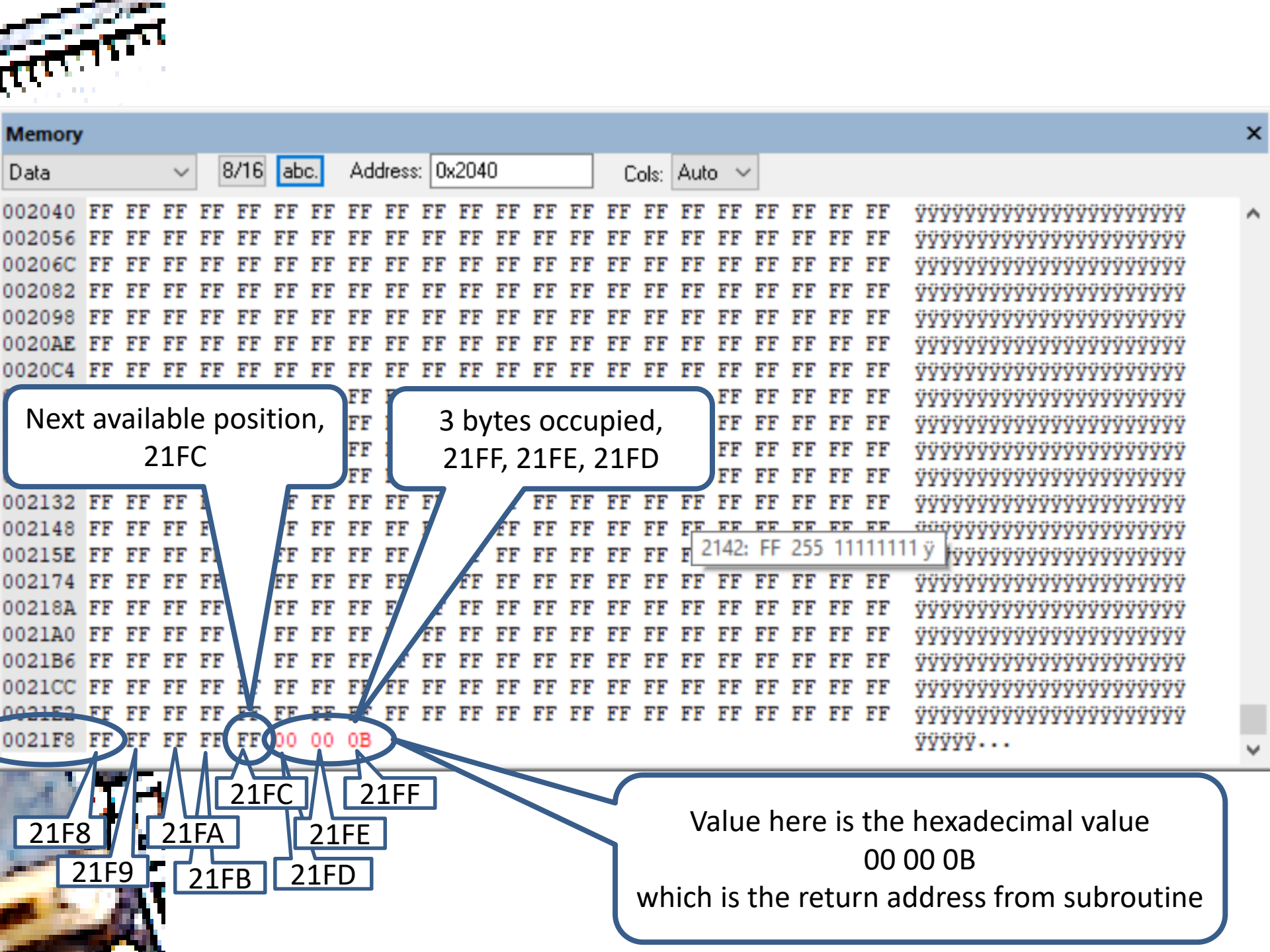


SP, Stack Pointer = 0x21FF

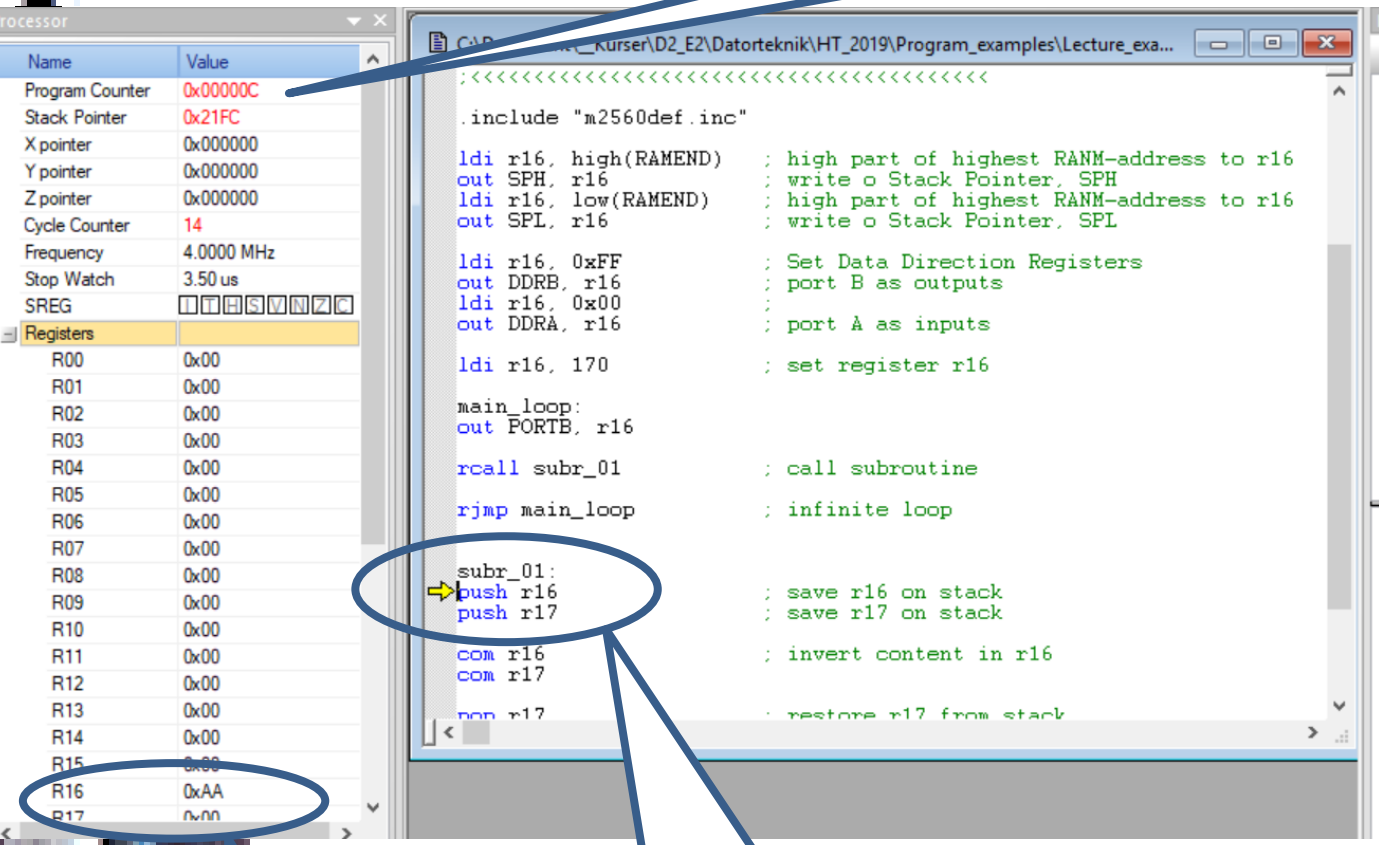
PC, Proogram Counter =
0x 00 00 0C



SP, Stack Pointer
= 0x21FC

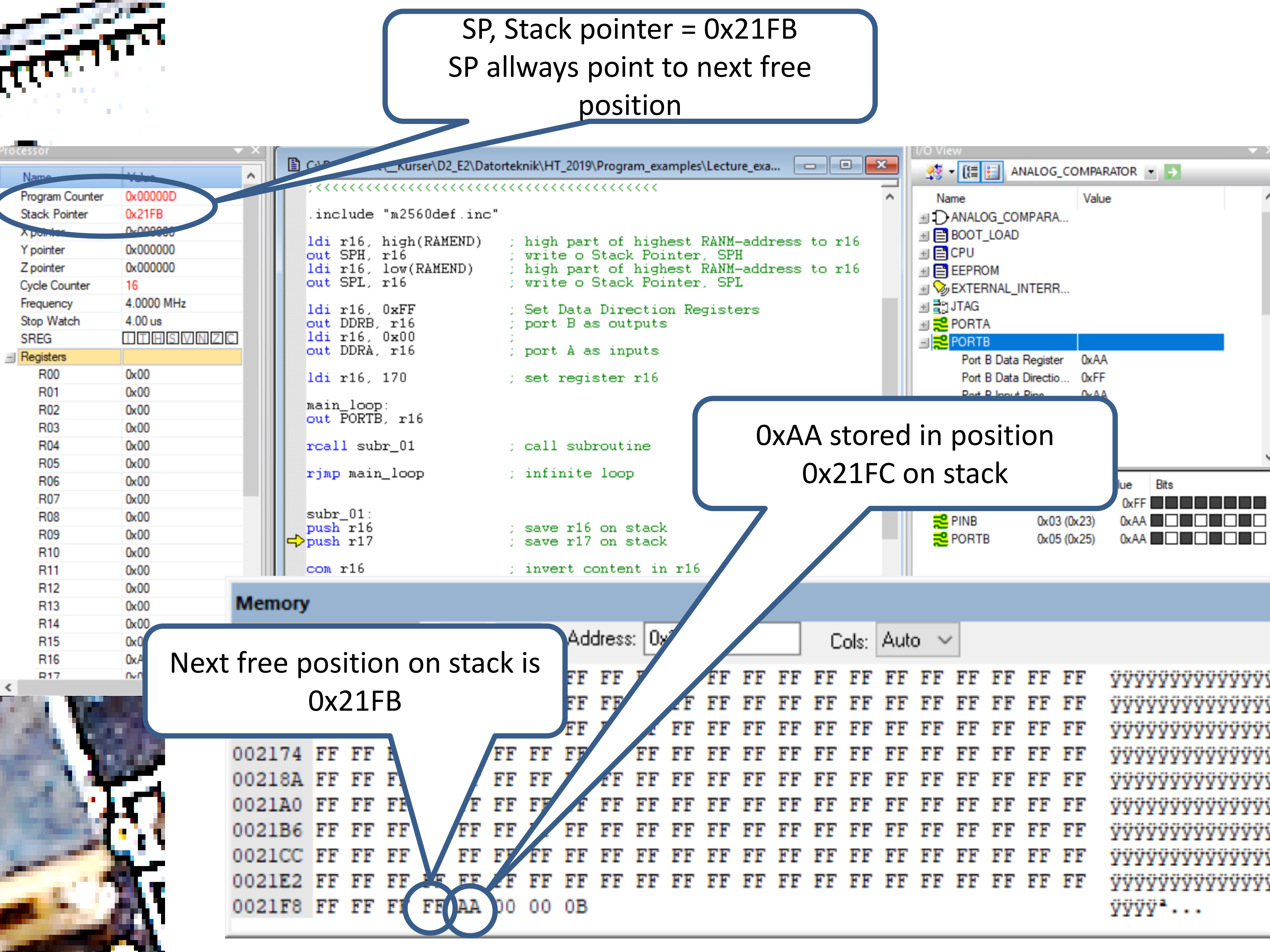

Memory
Data
8/16
abc.
Address: 0x2040
Cols: Auto
002040 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
002056 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00206C FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
002082 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
002098 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0020AE FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0020C4 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
002132 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
002148 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00215E FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
002174 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00218A FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0021A0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0021B6 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0021CC FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0021E2 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0021F8 FF FF FF FF FF 00 00 0B
2142: FF 255 11111111
Next available position, 21FC
3 bytes occupied, 21FF, 21FE, 21FD
21F8
21F9
21FA
21FB
21FC
21FD
21FE
21FF
Value here is the hexadecimal value
00 00 0B
which is the return address from subroutine

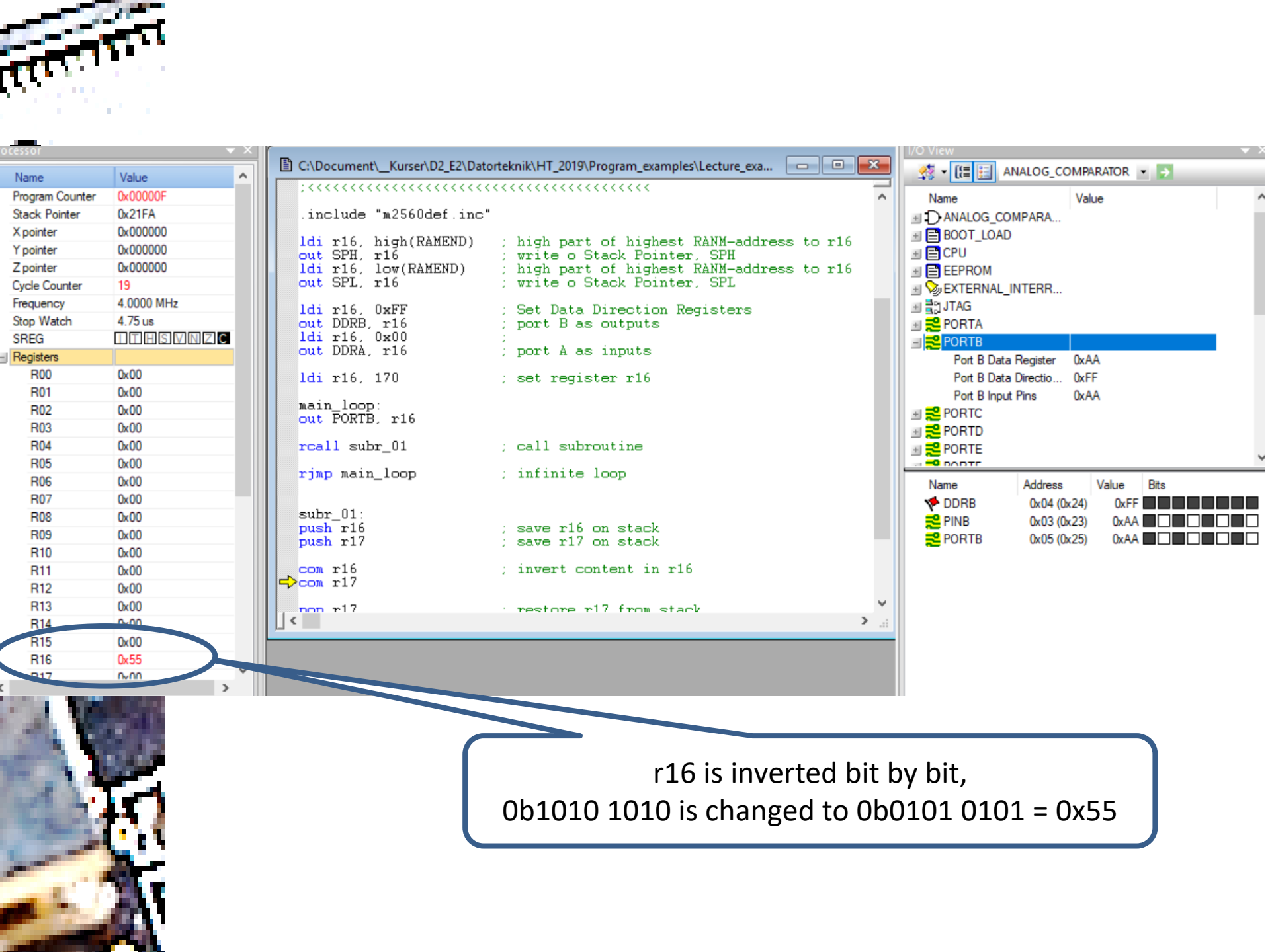
PC, Proogram Counter =
0x 00 00 0C



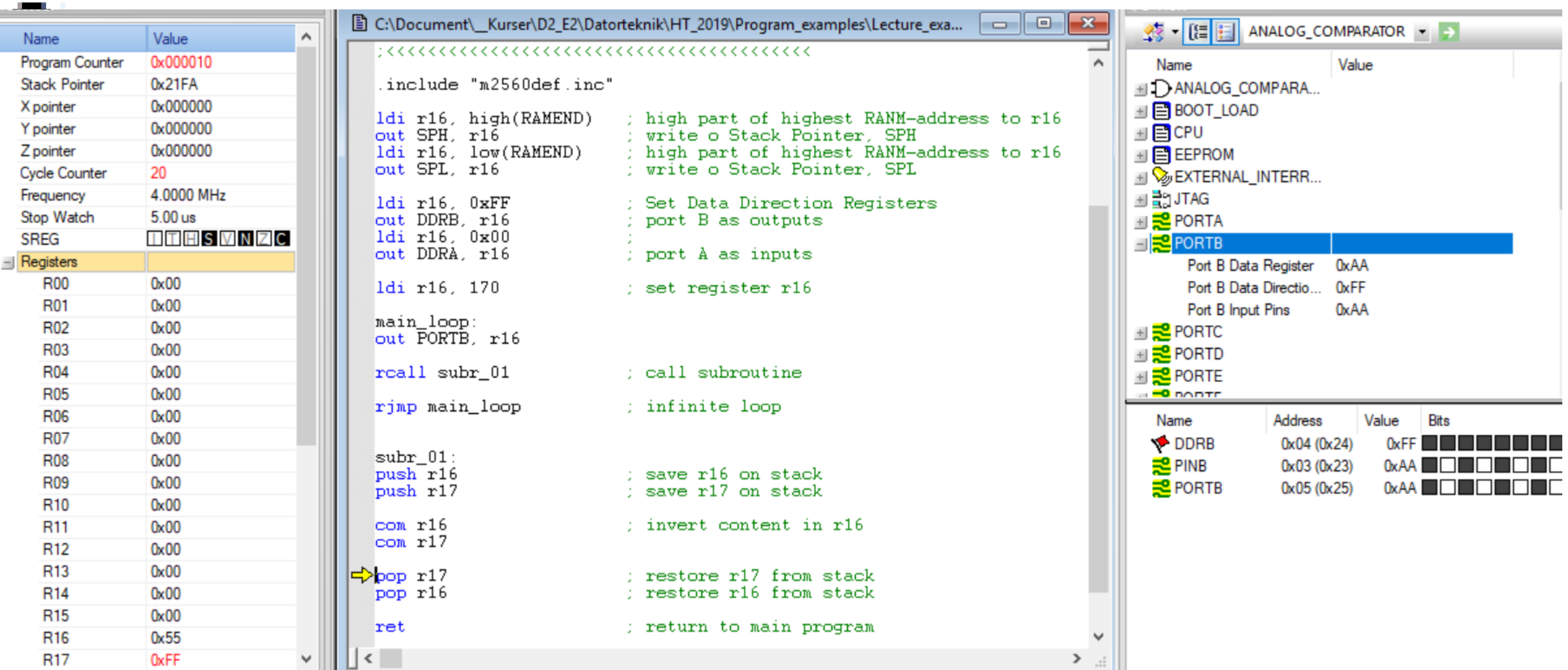
r16 = 0xAA
r17 = 0x00

push r16 will store the
content of r16 on the stack

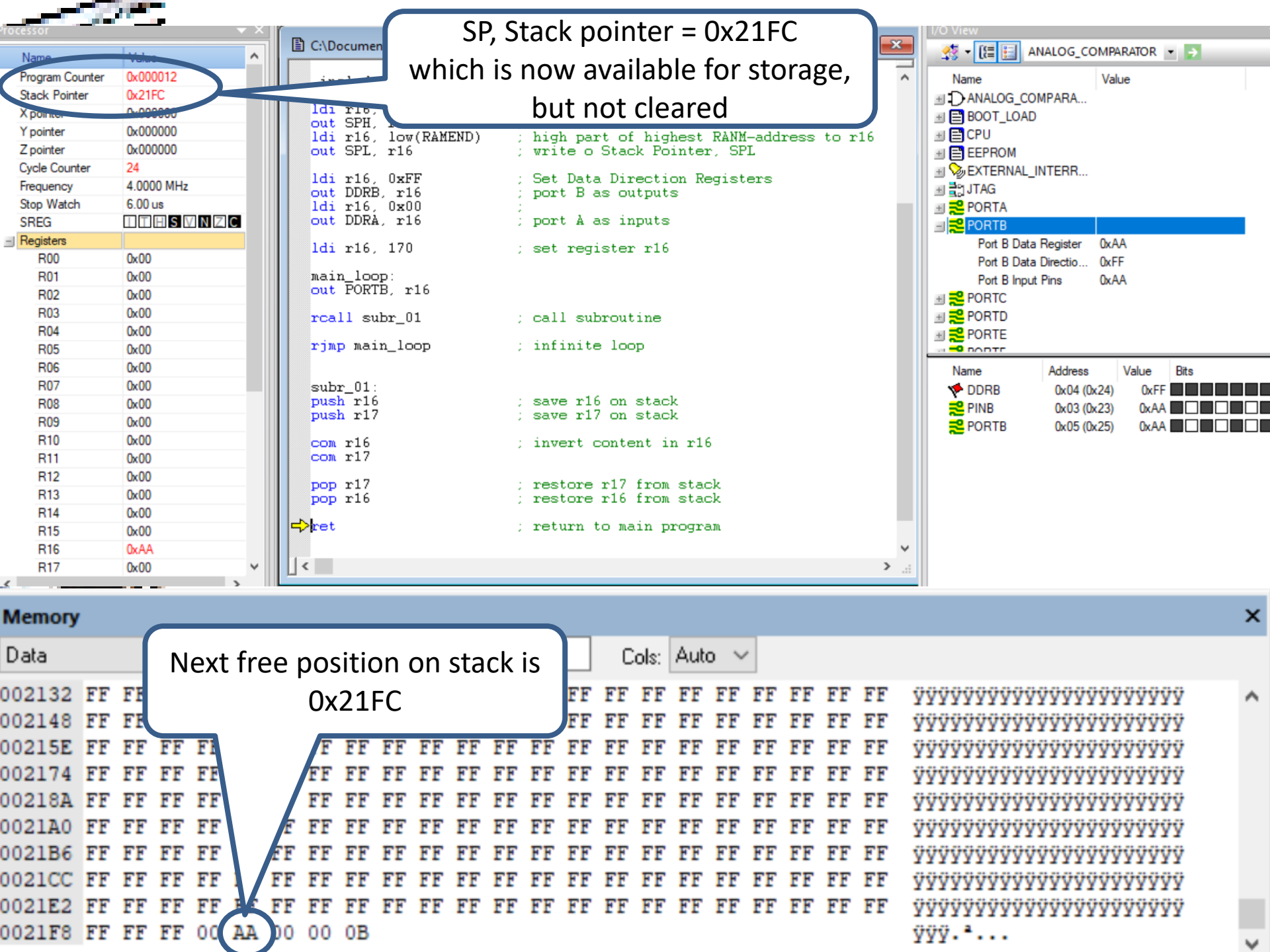

SP, Stack pointer = 0x21FB
SP allways point to next free position
0xAA stored in position 0x21FC on stack
Next free position on stack is 0x21FB
Processor
Program Counter 0x00000D
Stack Pointer 0x21FB
Y pointer 0x000000
Z pointer 0x000000
Cycle Counter 16
Frequency 4.0000 MHz
Stop Watch 4.00 us
SREG
Registers
.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
ldi r16, 0x00
out DDRA, r16 ; port A as inputs
ldi r16, 170 ; set register r16
main_loop:
out PORTB, r16
rcall subr_01 ; call subroutine
rjmp main_loop ; infinite loop
subr_01:
push r16 ; save r16 on stack
push r17 ; save r17 on stack
com r16 ; invert content in r16
I/O View
ANALOG_COMPARATOR
Port B Data Register 0xAA
Port B Data Directio... 0xFF
PINB 0x03 (0x23) 0xAA
PORTB 0x05 (0x25) 0xAA
Memory
Address:
Cols: Auto
0021F8 FF FF FF FF AA 00 00 0B

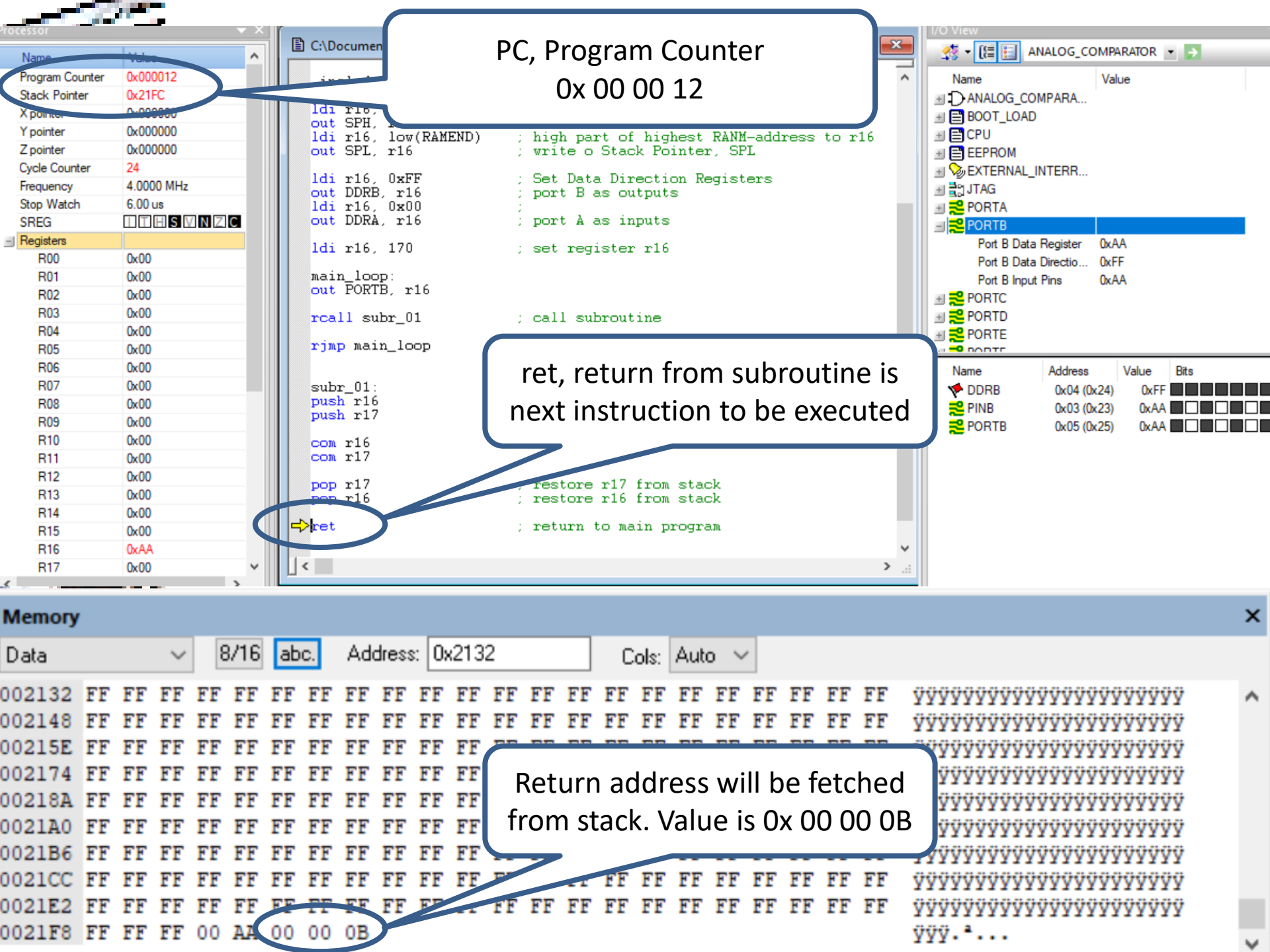


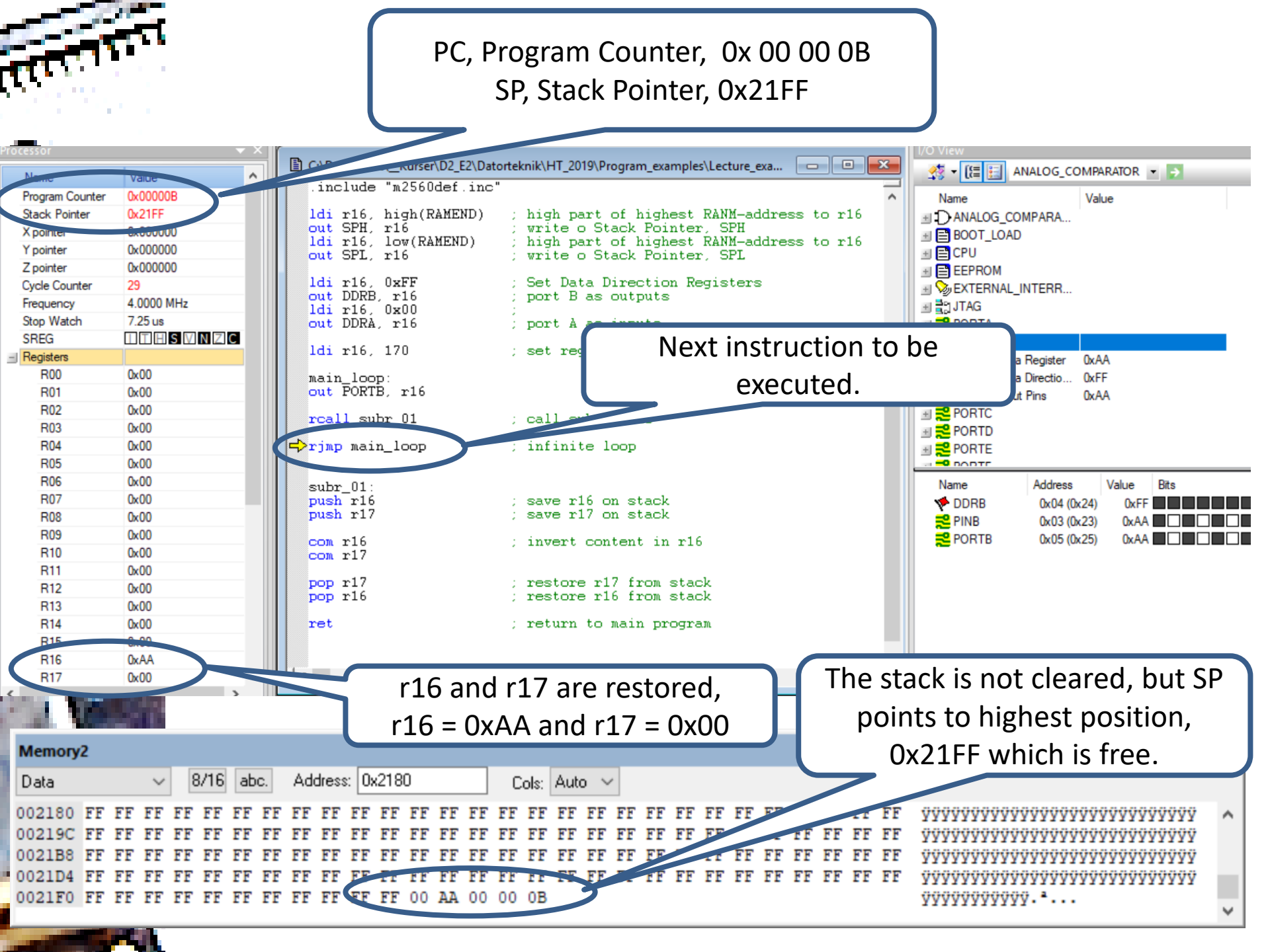
r16 is inverted bit by bit,
0b1010 1010 is changed to 0b0101 0101 = 0x55



Name
Value
Program Counter 0x000010
Stack Pointer 0x21FA
X pointer 0x000000
Y pointer 0x000000
Z pointer 0x000000
Cycle Counter 20
Frequency 4.0000 MHz
Stop Watch 5.00 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 0x55
R17 0xFF
C:\Document_Kurser\D2_E2\Datorteknik\HT_2019\Program_examples\Lecture_exa...
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
.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
ldi r16, 0x00 ;
out DDRA, r16 ; port A as inputs
ldi r16, 170 ; set register r16
main_loop:
out PORTB, r16
rcall subr_01 ; call subroutine
rjmp main_loop ; infinite loop
subr_01:
push r16 ; save r16 on stack
push r17 ; save r17 on stack
com r16 ; invert content in r16
com r17
pop r17 ; restore r17 from stack
pop r16 ; restore r16 from stack
ret ; return to main program
ANALOG_COMPARATOR
Name
Value
ANALOG_COMPARA...
BOOT_LOAD
CPU
EEPROM
EXTERNAL_INTERR...
JTAG
PORTA
PORTB
Port B Data Register 0xAA
Port B Data Directio... 0xFF
Port B Input Pins 0xAA
PORTC
PORTD
PORTE
Name
Address
Value
Bits
DDRB 0x04 (0x24) 0xFF
PINB 0x03 (0x23) 0xAA
PORTB 0x05 (0x25) 0xAA


SP, Stack pointer = 0x21FC
which is now available for storage, but not cleared
Next free position on stack is 0x21FC
Program Counter 0x000012
Stack Pointer 0x21FC
X pointer
Y pointer 0x000000
Z pointer 0x000000
Cycle Counter 24
Frequency 4.0000 MHz
Stop Watch 6.00 us
SREG
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 0xAA
R17 0x00
ldi r16, low(RAMEND) ; high part of highest RAM-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
ldi r16, 0x00 ;
out DDRA, r16 ; port A as inputs
ldi r16, 170 ; set register r16
main_loop:
out PORTB, r16
rcall subr_01 ; call subroutine
rjmp main_loop ; infinite loop
subr_01:
push r16 ; save r16 on stack
push r17 ; save r17 on stack
com r16 ; invert content in r16
com r17
pop r17 ; restore r17 from stack
pop r16 ; restore r16 from stack
ret ; return to main program
I/O View
ANALOG_COMPARATOR
Name Value
ANALOG_COMPARA...
BOOT_LOAD
CPU
EEPROM
EXTERNAL_INTERR...
JTAG
PORTA
PORTB
Port B Data Register 0xAA
Port B Data Directio... 0xFF
Port B Input Pins 0xAA
PORTC
PORTD
PORTE
Name Address Value Bits
DDRB 0x04 (0x24) 0xFF
PINB 0x03 (0x23) 0xAA
PORTB 0x05 (0x25) 0xAA
Memory
Data
Cols: Auto
002132 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
002148 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00215E FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
002174 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00218A FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0021A0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0021B6 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0021CC FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0021E2 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0021F8 FF FF FF 00 AA 00 00 0B


PC, Program Counter
0x 00 00 12
ret, return from subroutine is next instruction to be executed
Return address will be fetched from stack. Value is 0x 00 00 0B
Processor
Name
Value
Program Counter
0x000012
Stack Pointer
0x21FC
X pointer
0x000000
Y pointer
0x000000
Z pointer
0x000000
Cycle Counter
24
Frequency
4.0000 MHz
Stop Watch
6.00 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 0xAA
R17 0x00
C:\Documen
ldi r16, low(RAMEND) ; high part of highest RAM-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
ldi r16, 0x00 ;
out DDRA, r16 ; port A as inputs
ldi r16, 170 ; set register r16
main_loop:
out PORTB, r16
rcall subr_01 ; call subroutine
rjmp main_loop
subr_01:
push r16
push r17
com r16
com r17
pop r17 ; restore r17 from stack
pop r16 ; restore r16 from stack
ret ; return to main program
I/O View
ANALOG_COMPARATOR
Name
Value
ANALOG_COMPARA...
BOOT_LOAD
CPU
EEPROM
EXTERNAL_INTERR...
JTAG
PORTA
PORTB
Port B Data Register 0xAA
Port B Data Directio... 0xFF
Port B Input Pins 0xAA
PORTC
PORTD
PORTE
Name Address Value Bits
DDRB 0x04 (0x24) 0xFF
PINB 0x03 (0x23) 0xAA
PORTB 0x05 (0x25) 0xAA
Memory
Data
8/16
abc.
Address: 0x2132
Cols: Auto
002132 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
002148 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00215E FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
002174 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00218A FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0021A0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0021B6 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0021CC FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0021E2 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0021F8 FF FF FF 00 AA 00 00 0B


PC, Program Counter, 0x 00 00 0B
SP, Stack Pointer, 0x21FF
Next instruction to be executed.
r16 and r17 are restored,
r16 = 0xAA and r17 = 0x00
The stack is not cleared, but SP points to highest position, 0x21FF which is free.
Program Counter 0x00000B
Stack Pointer 0x21FF
X pointer 0x000000
Y pointer 0x000000
Z pointer 0x000000
Cycle Counter 29
Frequency 4.0000 MHz
Stop Watch 7.25 us
SREG
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
R16 0xAA
R17 0x00
.include "m2560def.inc"
ldi r16, high(RAMEND) ; high part of highest RAM-address to r16
out SPH, r16 ; write o Stack Pointer, SPH
ldi r16, low(RAMEND) ; high part of highest RAM-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
ldi r16, 0x00 ;
out DDRA, r16 ; port A as inputs
ldi r16, 170 ; set reg
main_loop:
out PORTB, r16
rcall subr_01 ; call sub
rjmp main_loop ; infinite loop
subr_01:
push r16 ; save r16 on stack
push r17 ; save r17 on stack
com r16 ; invert content in r16
com r17
pop r17 ; restore r17 from stack
pop r16 ; restore r16 from stack
ret ; return to main program
ANALOG_COMPARATOR
Name Value
ANALOG_COMPARA...
BOOT_LOAD
CPU
EEPROM
EXTERNAL_INTERR...
JTAG
PORTC
PORTD
PORTE
Register 0xAA
Directio... 0xFF
Pins 0xAA
Name Address Value Bits
DDRB 0x04 (0x24) 0xFF
PINB 0x03 (0x23) 0xAA
PORTB 0x05 (0x25) 0xAA
Memory2
Data
8/16
abc.
Address: 0x2180
Cols: Auto
002180 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00219C FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0021B8 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0021D4 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0021F0 FF FF FF FF FF FF FF FF FF FF 00 AA 00 00 0B

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) ← 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:
(i)	PUSH Rr	0 ≤ r ≤ 31

Program Counter: PC ← PC + 1

Stack: 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:	Syntax:	Operands:	Program Counter:	Stack:
(i)	Rd ← STACK	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)