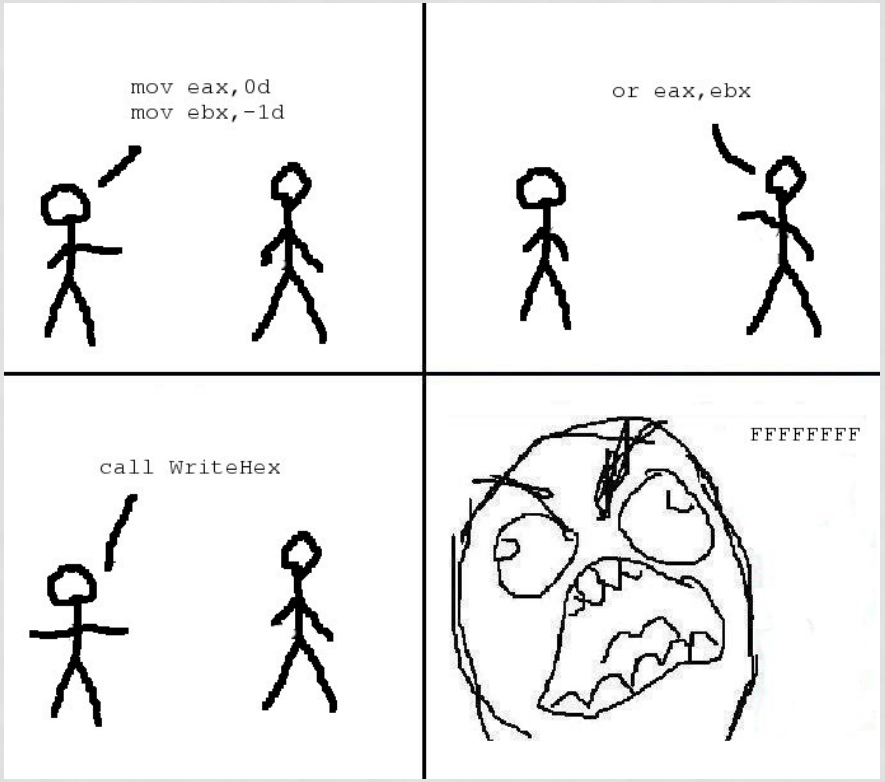


ASSEMBLY & C

WEEK 1-1

AGENDA

week	onderwerp	week	week
1	de structuur van AVR-assembly AVR instructies AVR registers en I/O ATmega memory map Atmel Studio AVR expressies en directives AVR addressing modes	3	de structuur van C-programma's ATMEL studio en AVR libc typen, constanten en operatoren AVR register access in C control statements functies & stackframe visibility scope arrays & strings struct & enum
2	flow of control spring instructies, control structuren Arduino UNO AVR studio stack & subroutines interrupts timer/counters switch bounce	4	interrupts in C TM1638 led&key UART PWM & ADC using a TTC-scheduler state diagram

AGENDA

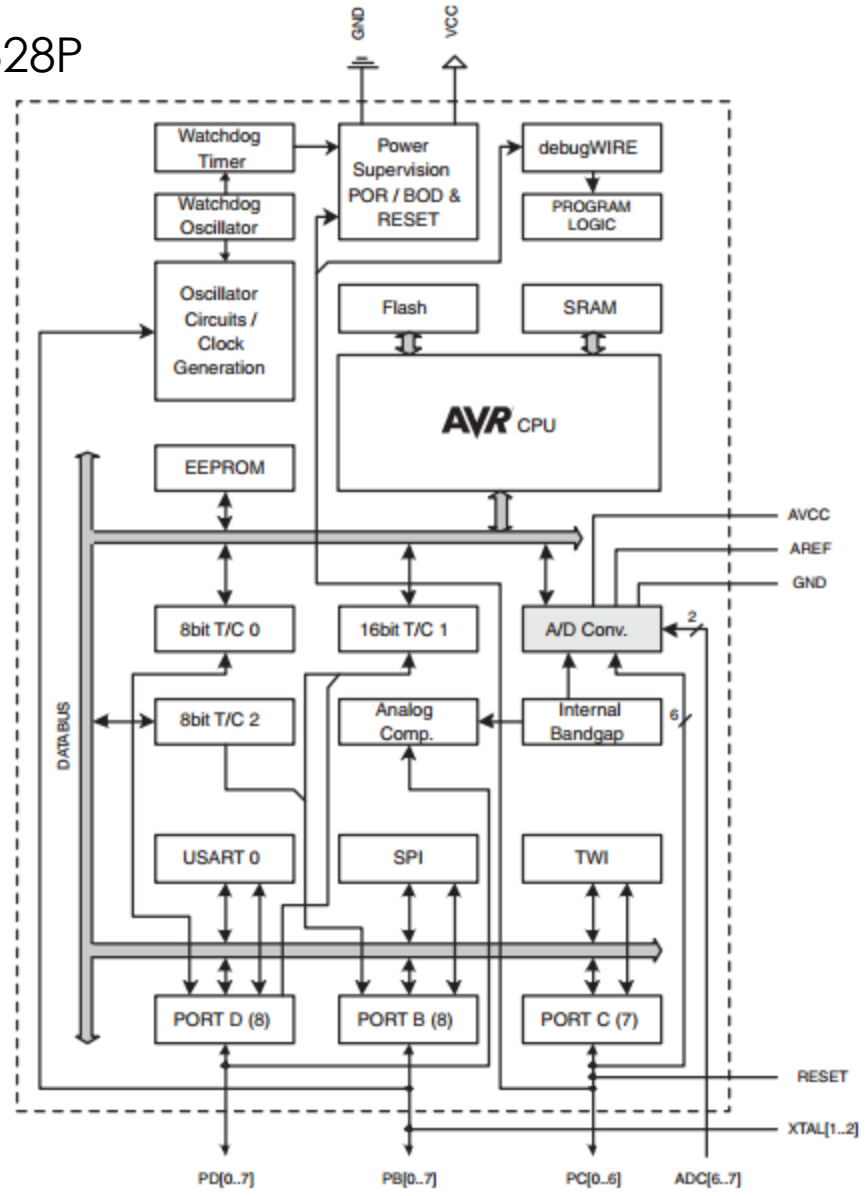
- **intro**
- de structuur van AVR assembly
- AVR instructies
- AVR registers
- I/O poorten
- ATMega memory map
- ATMEL studio

AVR

- Atmel AVR 8-bit MCU's zijn er in diverse configuraties en behuizingen
  - tinyAVR, MegaAVR, XMAGA, USB AVR, Automotive AVR, ...
  - zelfde CPU, andere peripherals
  - zelfde instructie set en register set
- wij gebruiken ATmega32 (ATmega328P)

ATmega328P

Figure 2-1. Block Diagram



ATMEGA328P SPECS

- 131 instructies
- 32 8-bit GP registers
- up to 20 MIPS throughput
- 32K programmable flash (code)
- 2K interne SRAM
- 1K EEPROM
- peripherals : timer/counters, seriële & parallelle I/O, ADC

ATMEL 8-BIT MICROCONTROLLER WITH 4/8/16/32KBYTES IN-SYSTEM PROGRAMMABLE FLASH

DATASHEET

Features

- High Performance, Low Power Atmel®AVR® 8-Bit Microcontroller Family
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 4/8/16/32KBytes of In-System Self-Programmable Flash program memory
  - 256/512/512/1KBytes EEPROM
  - 512/1K/1K/2KBytes Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C(1)
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
- Atmel® QTouch® library support
  - Capacitive touch buttons, sliders and wheels
  - QTouch and QMatrix® acquisition
  - Up to 64 sense channels
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode

AGENDA

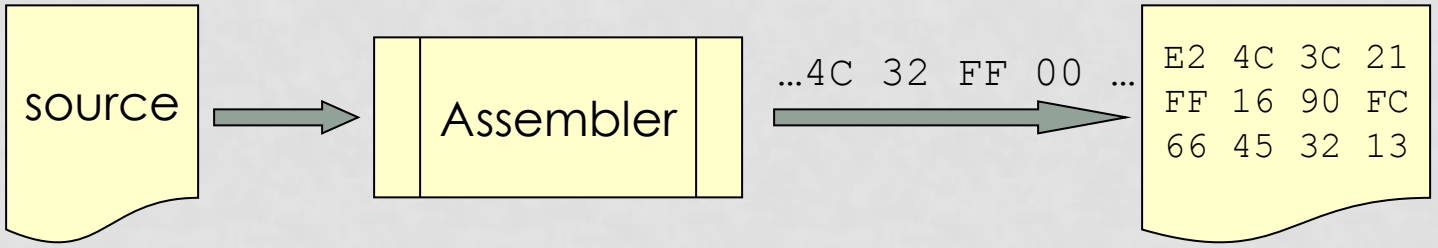
- intro
- **de structuur van AVR assembly**
- AVR instructies
- AVR registers
- I/O poorten
- ATMega memory map
- ATMEL studio

AVR ASSEMBLY

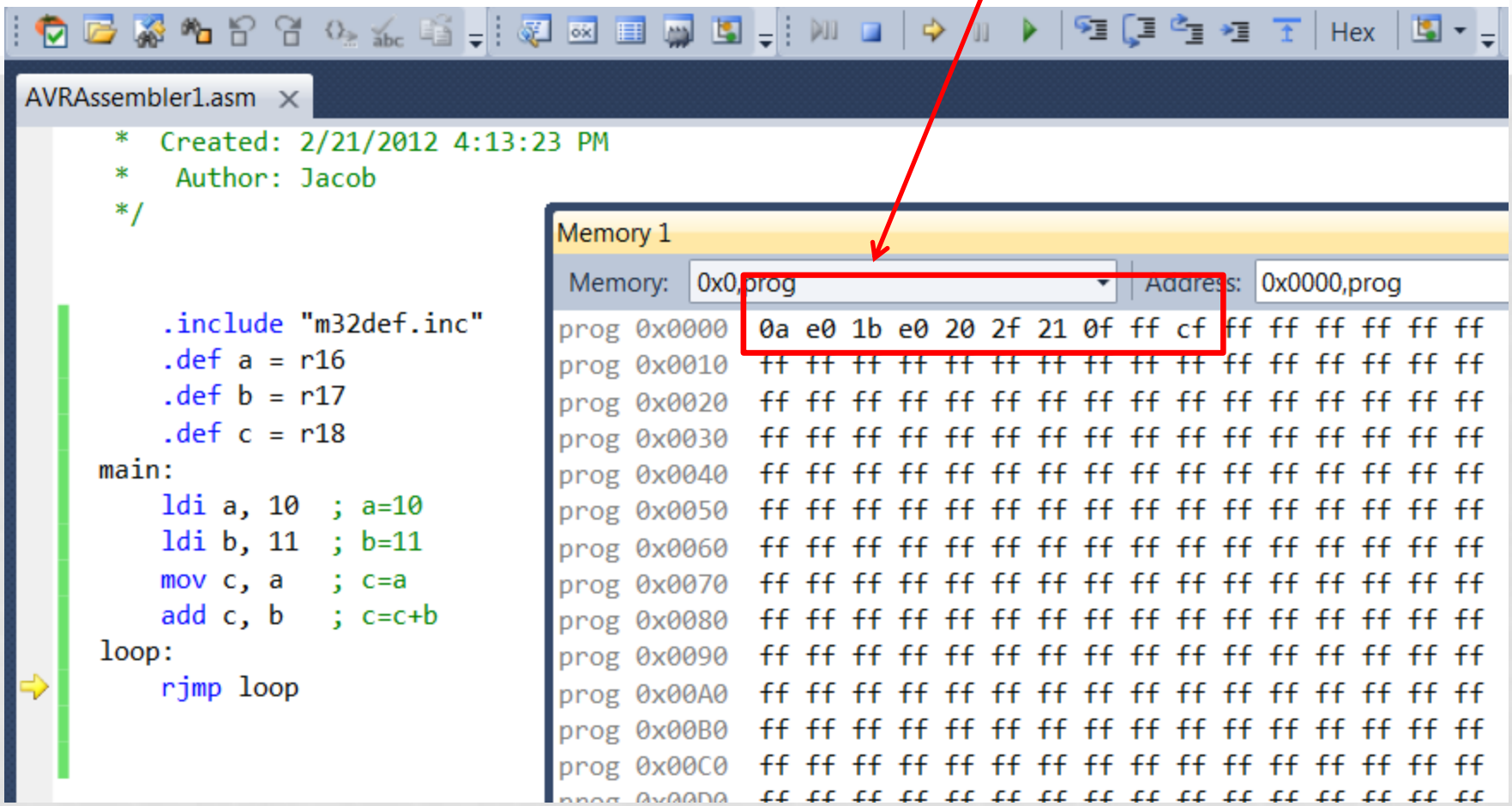
- elke CPU-architectuur eigen instructieset
  - AVR assembly : MCU's uit Atmel AVR familie
- assembly statements : machine instructies = 1:1
  - assembler vertaalt source code (tekst) naar machine instructies (bits)
- ADD R1, R2
  - mnemonic ADD : leesbare instructies
  - operanden R1 en R2 : data waarmee de instructie moet werken

ASSEMBLER

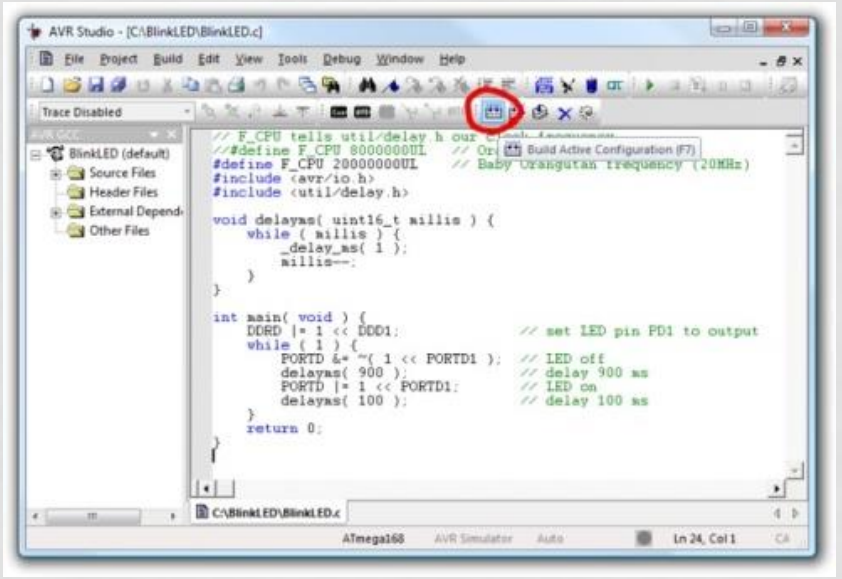
- assembler vertaalt source code (tekst) naar machine instructies (bits)



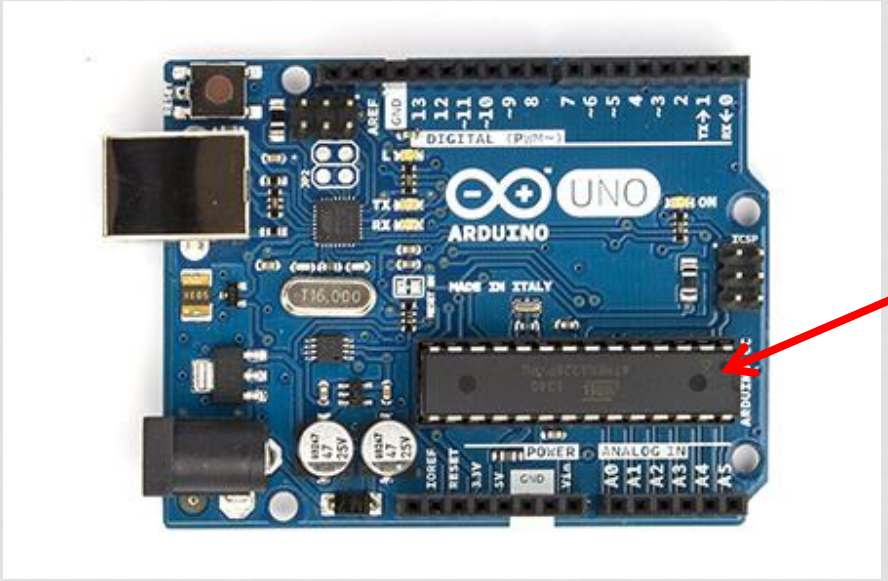
5 instructies



ATmega238P simulator op PC
(zit in AVR studio)



ATmega238P op Arduino UNO



WAAROM ASSEMBLY LEREN ?

- wat doet de CPU
- wat doet een compiler, linker of interpreter
- wat is een VM
- wat is een proces
- hoe werkt multi-threading
- wat is een interrupt
- hacker trucs ... en wat is onveilige code
- hoe werkt code optimalisatie
- wat is een dis-assembler

AVR ASSEMBLY

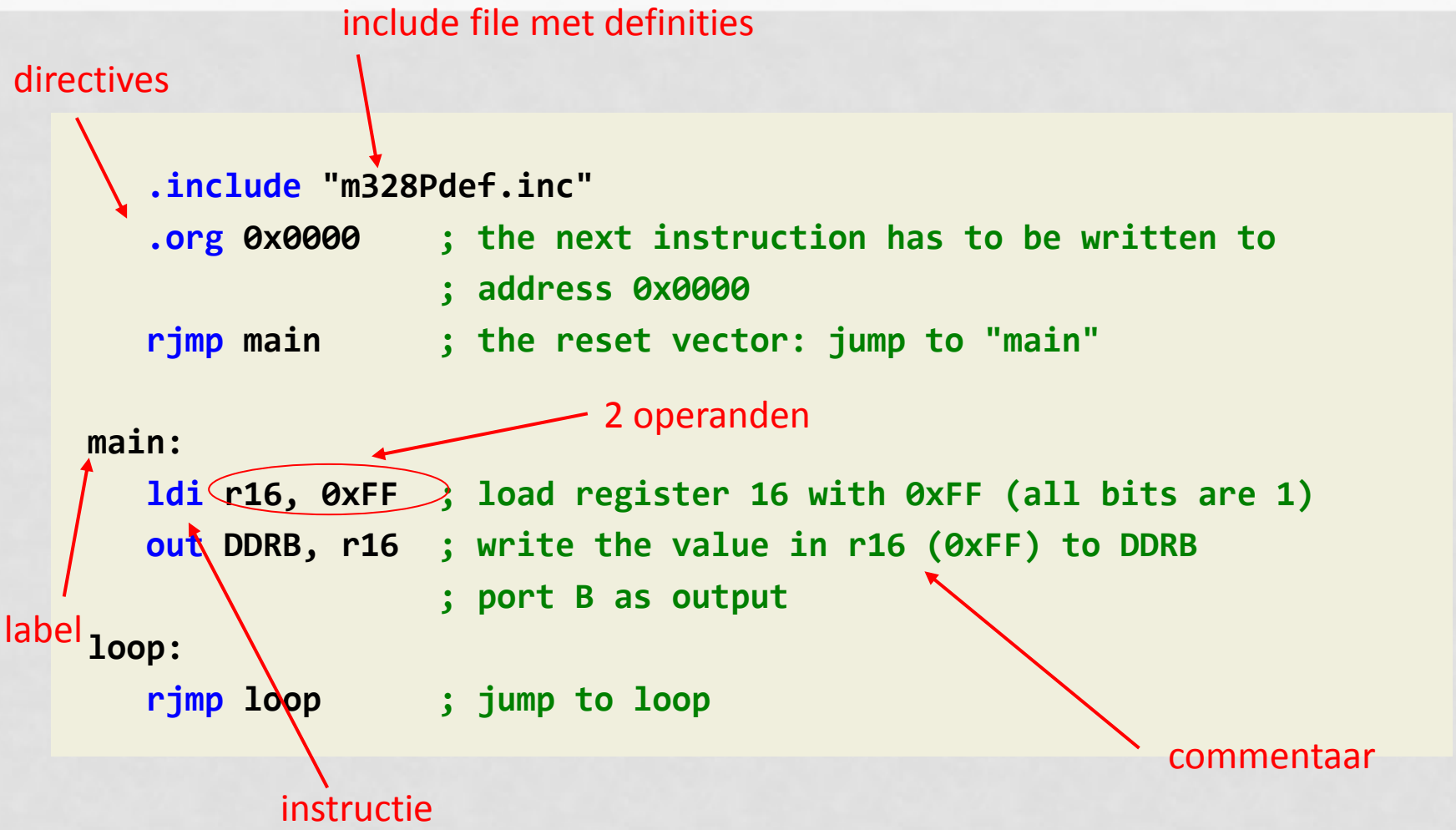
- syntax :

- `[label:] directive [operands] [;comment]`
- `[label:] instruction [operands][;comment]`

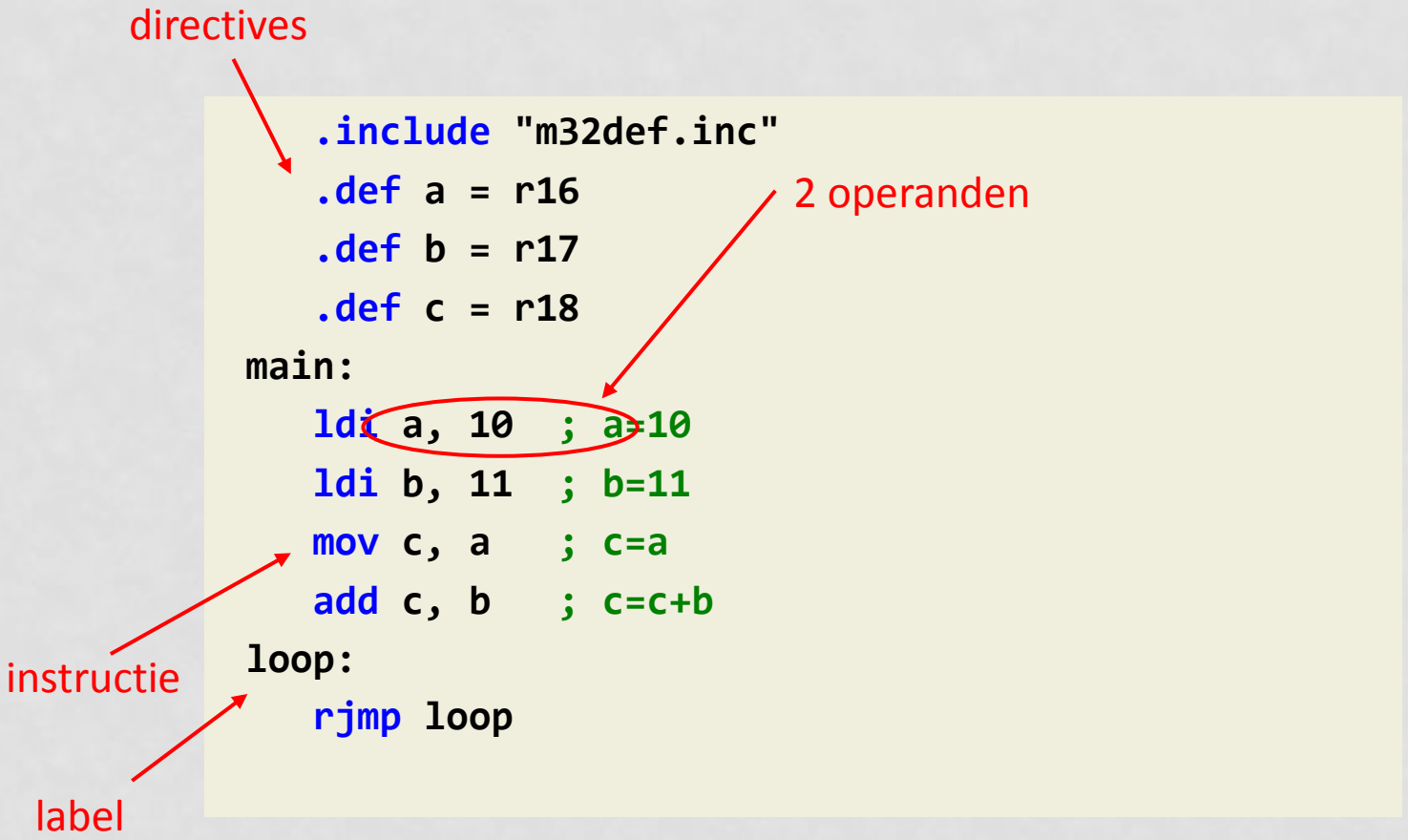
- labels representeren adressen
  - labels in het code segment zijn 16 bit
  - labels in het data segment zijn 8 bit

- wat is het verschil tussen assembler directive en instructie ?

VOORBEELD 1


include file met definities
directives
.include "m328Pdef.inc"
.org 0x0000 ; the next instruction has to be written to
; address 0x0000
rjmp main ; the reset vector: jump to "main"
2 operanden
main:
ldi r16, 0xFF ; load register 16 with 0xFF (all bits are 1)
out DDRB, r16 ; write the value in r16 (0xFF) to DDRB
; port B as output
label
loop:
rjmp loop ; jump to loop
commentaar
instructie

VOORBEELD 2


directives
2 operanden
instructie
label
.include "m32def.inc"
.def a = r16
.def b = r17
.def c = r18
main:
ldi a, 10 ; a=10
ldi b, 11 ; b=11
mov c, a ; c=a
add c, b ; c=c+b
loop:
rjmp loop

COMMENTAAR

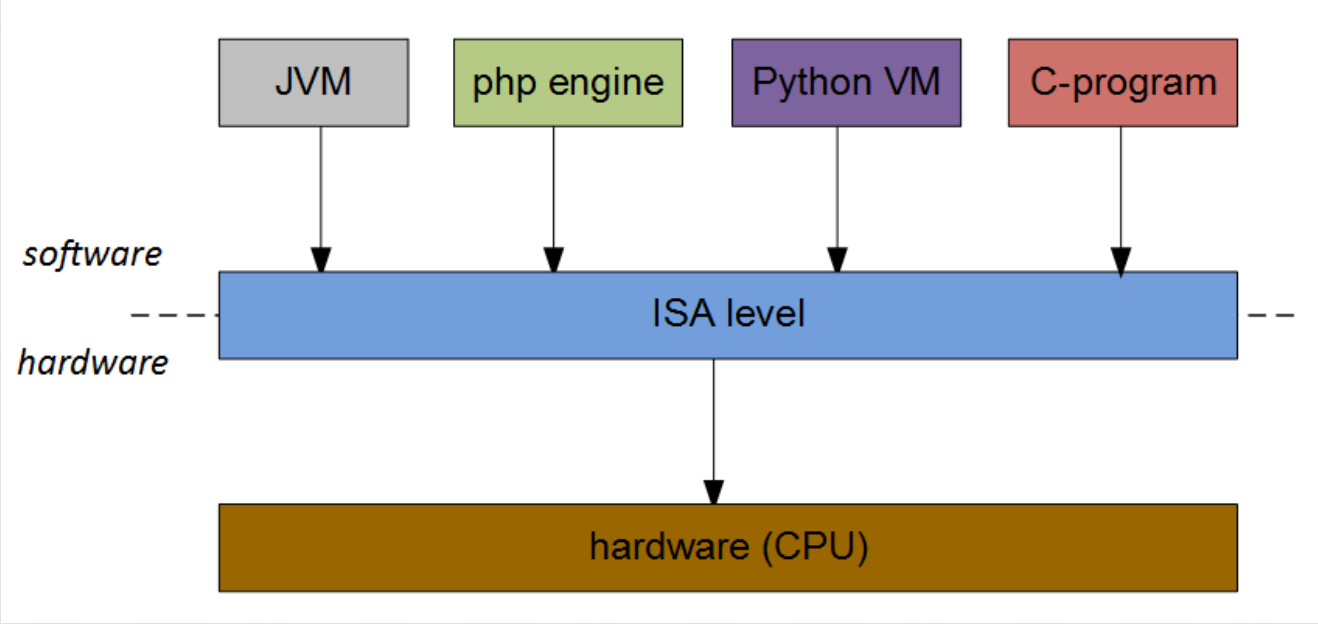
- past op een regel
  - begin met ; tot einde regel
- blok
  - tussen /*en */

AGENDA

- intro
- de structuur van AVR assembly
- **AVR instructies**
- AVR registers
- I/O poorten
- ATMega memory map
- ATMEL studio

ISA

- Instruction Set Architecture
- is het interface tussen hardware en software
- geeft aan wat de programmeur kan (functionele specificatie)



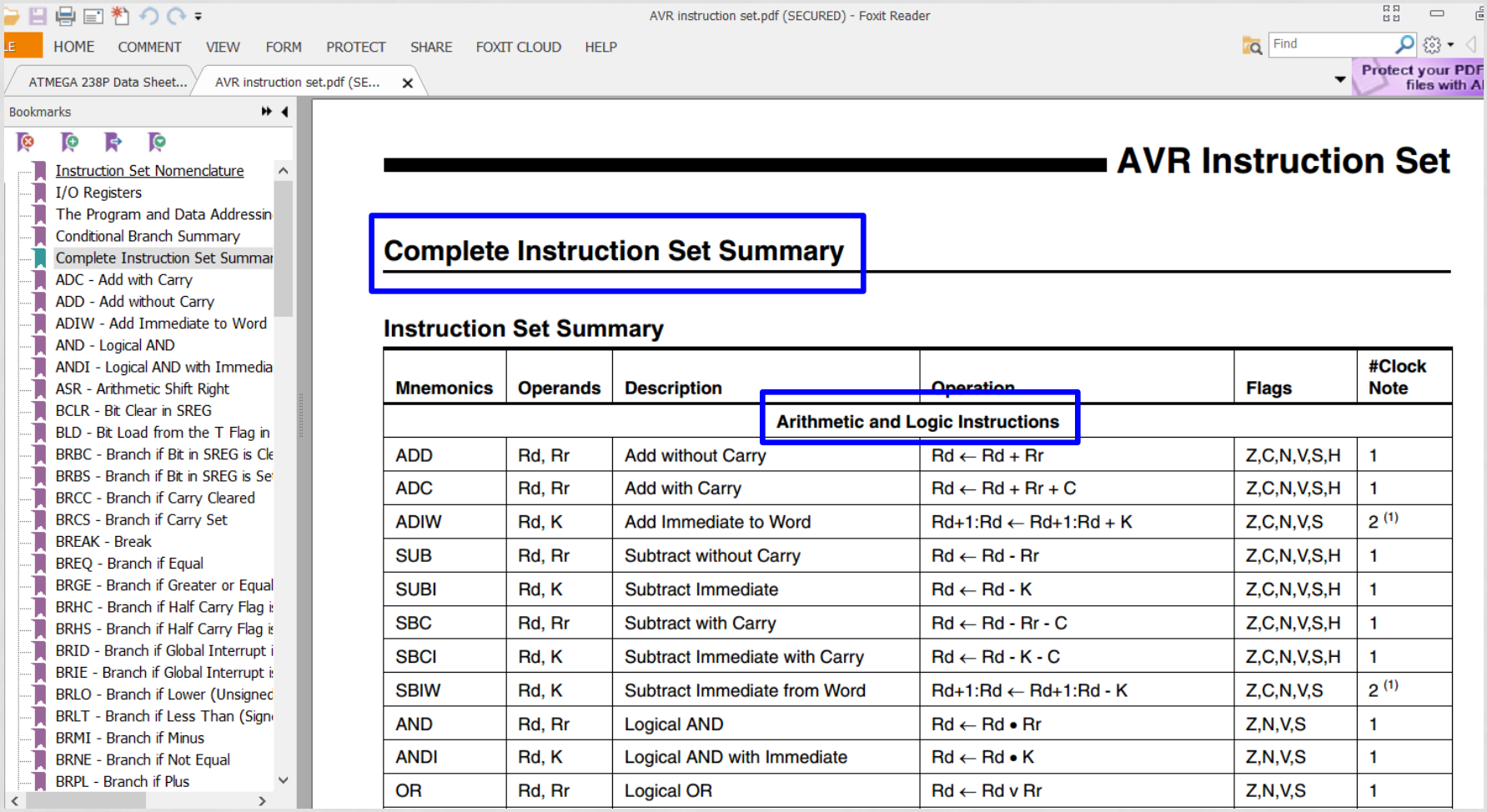
AVR INSTRUCTIES

- alle instructies zijn 16 *of* 32 bit
  - 32 bit : CALL, JMP, LDS, STS
- meeste instructies zijn 16 bit en duren 1 klokcyclus

TYPEN INSTRUCTIES

- data transfer (18)
  - LDI load immediate
- arithmetic and logic (28)
  - ADD add without carry
- branch (sprong) (38)
  - JMP jump
- bit and bit-test (28)
  - LSL logical shift left
- MCU control (4)
  - NOP en SLEEP

AVR INSTRUCTIESET



LDI – Load Immediate

Description:

Loads an 8 bit constant directly to register 16 to 31.

immediate : data als literal meegeven

	Operation:		
(i)	Rd ← K		
	Syntax:	**Operands:**	**Program Counter:**
(i)	LDI Rd,K	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	PC ← PC + 1

16-bit Opcode:

1110	KKKK	dddd	KKKK

8 bit constante

instructie formaat (16 bit)

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
clr   r31       ; Clear Z high byte
ldi   r30,$F0   ; Set Z low byte to $F0
lpm             ; Load constant from Program
                ; memory pointed to by Z
```

Words: 1 (2 bytes)

Cycles: 1

in 1 CPU cycle

ADD – Add without Carry

d.w.z. zonder carry ***als input***

Description:

Adds two registers without the C Flag and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd + Rr

resultaat in Rd

	Syntax:	Operands:	Program Counter:
(i)	ADD Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0000	11rd	dddd	rrrr

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H: $Rd3 \bullet Rr3 + Rr3 \bullet \overline{R3} + \overline{R3} \bullet Rd3$
Set if there was a carry from bit 3; cleared otherwise

S: N ⊕ V, For signed tests.

V: $Rd7 \bullet Rr7 \bullet \overline{R7} + \overline{Rd7} \bullet \overline{Rr7} \bullet R7$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

C: $Rd7 \bullet Rr7 + Rr7 \bullet \overline{R7} + \overline{R7} \bullet Rd7$
Set if there was carry from the MSB of the result; cleared otherwise.

carry flag wordt gezet

R (Result) equals Rd after the operation.

Example:

```
add   r1,r2     ; Add r2 to r1 (r1=r1+r2)
add   r28,r28   ; Add r28 to itself (r28=r28+r28)
```

resultaat in r1

Words: 1 (2 bytes)
Cycles: 1

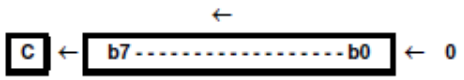
LSL – Logical Shift Left

Description:

Shifts all bits in Rd one place to the left. Bit 0 is cleared. Bit 7 is loaded into the C Flag of the SREG. This operation effectively multiplies signed and unsigned values by two.

Operation:

(i)



een "0" inschuiven

	Syntax:	Operands:	Program Counter:
(i)	LSL Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode: (see ADD Rd,Rd)

0000	11dd	dddd	dddd

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H: Rd3

S: N ⊕ V, For signed tests.

V: N ⊕ C (For N and C after the shift)

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

C: Rd7
Set if, before the shift, the MSB of Rd was set; cleared otherwise.

bit 7 wordt verplaatst naar de carry flag

R (Result) equals Rd after the operation.

Example:

```
add     r0,r4     ; Add r4 to r0
lsl     r0        ; Multiply r0 by 2
```

Words: 1 (2 bytes)
Cycles: 1

LDS – Load Direct from Data Space

Description:

Loads one byte from the data space to a register. For parts with SRAM, the data space consists of the Register File, I/O memory and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the register file only. The EEPROM has a separate address space.

A 16-bit address must be supplied. Memory access is limited to the current data segment of 64K bytes. The LDS instruction uses the RAMPD Register to access memory above 64K bytes. To access another data segment in devices with more than 64K bytes data space, the RAMPD in register in the I/O area has to be changed.

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

Operation:

(i) Rd ← (k)

	Syntax:	Operands:	Program Counter:
(i)	LDS Rd,k	0 ≤ d ≤ 31, 0 ≤ k ≤ 65535	PC ← PC + 2

32-bit Opcode:

1001	000d	dddd	0000
kkkk	kkkk	kkkk	kkkk

32 bit instructie formaat :
- 11 bit : opcode ID
- 5 bit : register
- 16 bit : adres

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
lds   r2,$FF00     ; Load r2 with the contents of data space location $FF00
add   r2,r1        ; add r1 to r2
sts   $FF00,r2     ; Write back
```

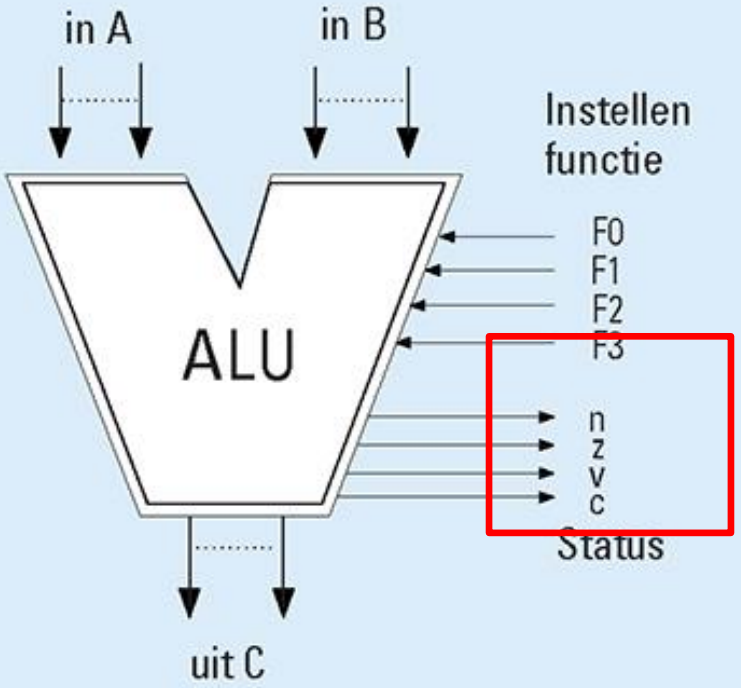
Words: 2 (4 bytes)
Cycles: 2

AGENDA

- intro
- de structuur van AVR assembly
- AVR instructies
- **AVR registers**
- I/O poorten
- ATMega memory map
- ATMEL studio

STATUS REGISTER

Figuur 3.7 **De ALU**



F3	F2	F1	F0	Functie
0	0	0	0	C = A
0	0	0	1	C = B
0	0	1	0	C = A + 1
0	0	1	1	C = A + B
0	1	0	0	C = A – B
0	1	0	1	C = A and B
0	1	1	0	C = A or B
..				..
etc.				

STATUS REGISTER

- bijna alle instructies werken het Status Register bij

naam	*functie*
I	Global Interrupt Enable
T	Bit Copy Storage
H	Half Carry Flag
S	Sign Bit
V	Two's Compliment Overflow Flag
N	Negative Flag
Z	Zero Flag
C	Carry Flag

ALGEMENE REGISTERS

- alle registers 8 bit
  - enige datatype is byte (8-bit)
  - voor 16-bit integer dus 2x lezen of schrijven
- tip : geef namen aan registers

```
.DEF temp=R16
```

- 32 "general purpose" registers :
  - R0..R15 : **geen** immediate adressering
  - R16..R31 : wel immediate adressering

```
ldi r16,30   ; r16=30
mov r0,r16   ; r0=r16
inc r0       ; r0++
add r0,r16   ; r0=r0+r16
```

```
ldi r15,30   ; r15=30
```

SPECIALE TOEPASSINGEN ALGEMENE REGISTERS

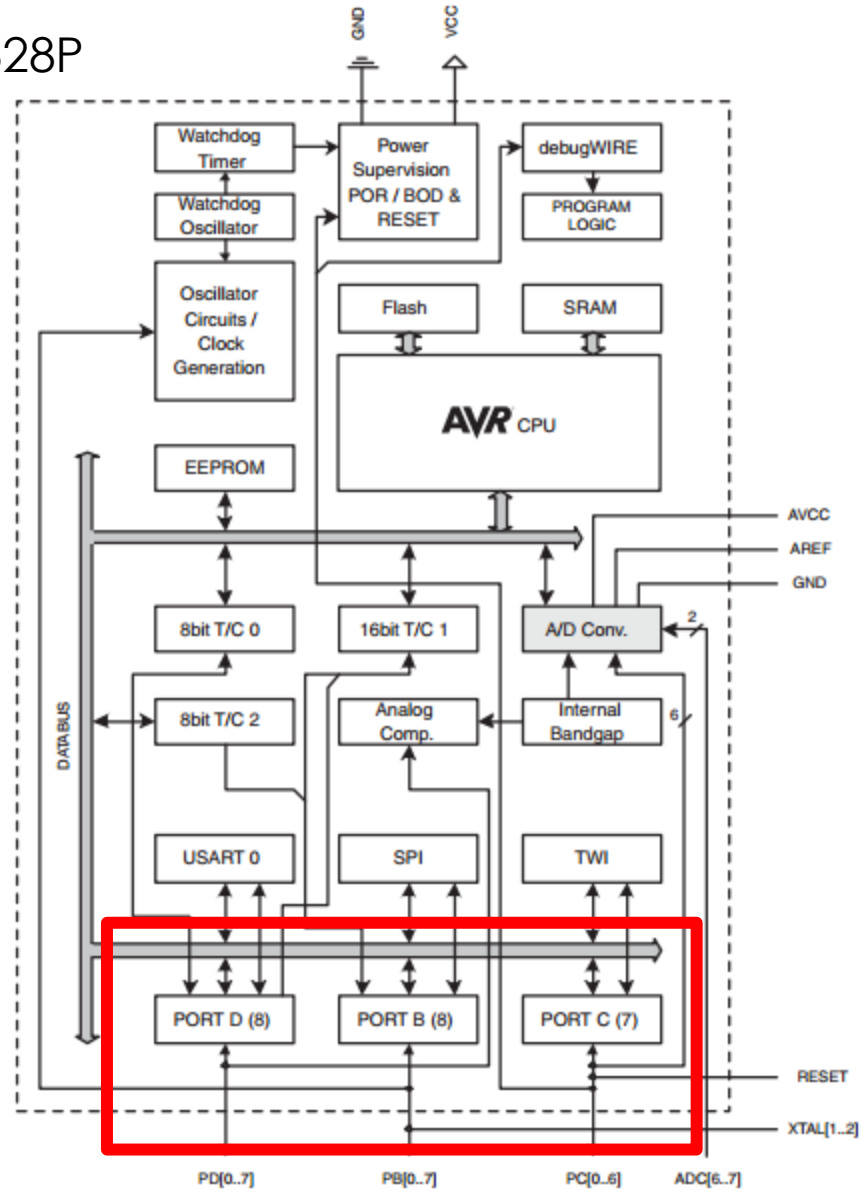
- resultaat MUL
  - R1:R0 store the result of multiplication
- resultaat LPM (zonder operanden)
  - R0 stores the data loaded from the program memory
- drie pointer registers :
  - X : R26:R27
  - Y : R28:R29
  - Z : R30:R31

AGENDA

- intro
- de structuur van AVR assembly
- AVR instructies
- AVR registers
- **I/O poorten**
- ATMega memory map
- ATMEL studio

Figure 2-1. Block Diagram

ATmega328P

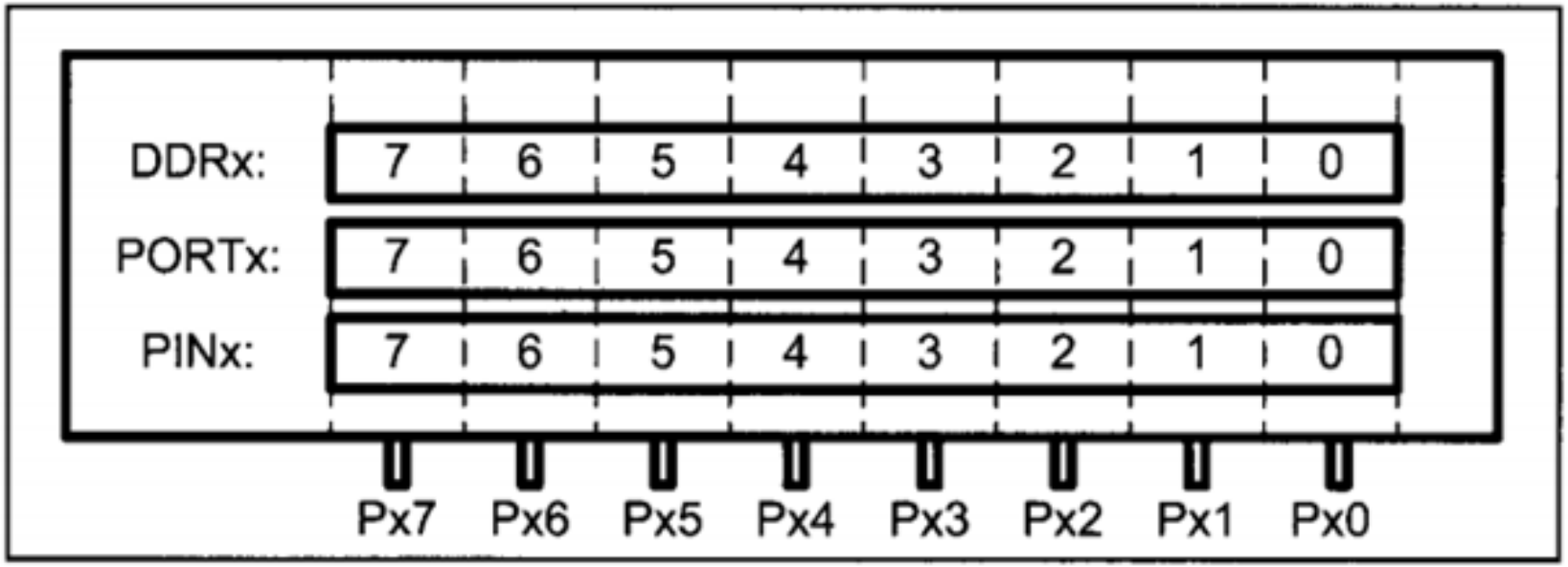


I/O POORTEN

- elke poort kan voor input en output zijn
- elke poort heeft 3 8-bit registers
- DDRB : instellen richting van poort B (input of output)
  - '1' betekent output, '0' betekent input
- PINB : lezen van input op poort B
- PORTB : schrijven van output naar poort B

I/O POORTEN

x = B,C of D

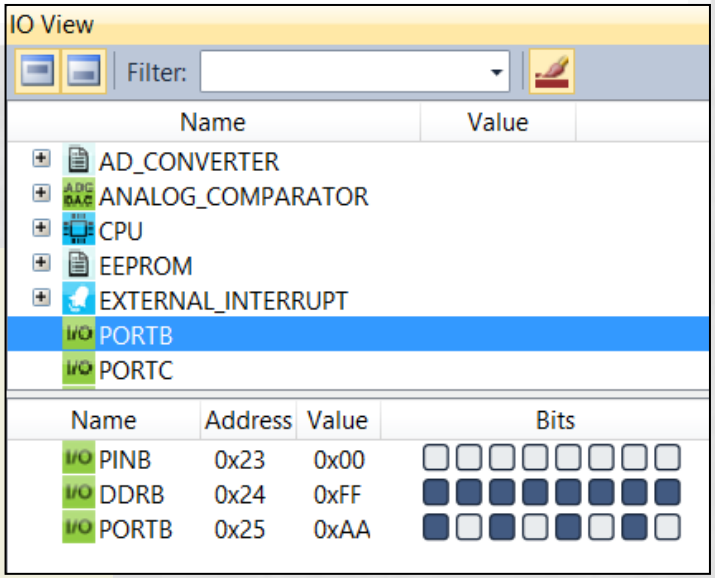


VOORBEELD 1

```
.include "m328Pdef.inc"

start:
   clr r16           ; r16=0x00
   out DDRB, r16     ; set port B as input
   in r16, PINB      ; read port B into r16

   ldi r16, 0x0F     ; r16=0x0F
   out DDRD, r16     ; set first four pins on Port D as
                     ; input and the others as utpu
   in r16, PIND      ; read port D into r16

end:
   rjmp end
```

VOORBEELD 2

```
.include "m328Pdef.inc"

start:
   ser r16              ; r16=0xFF
   out DDRB, r16        ; set port B as output
   ldi r16, 0xAA
   out PORTB, r16       ; write 0xAA to port B

   ldi r16, (1 << 3)    ; r16 = 0b00000100
   out DDRD, r16        ; set only PD2 as output
end:
   rjmp end
```

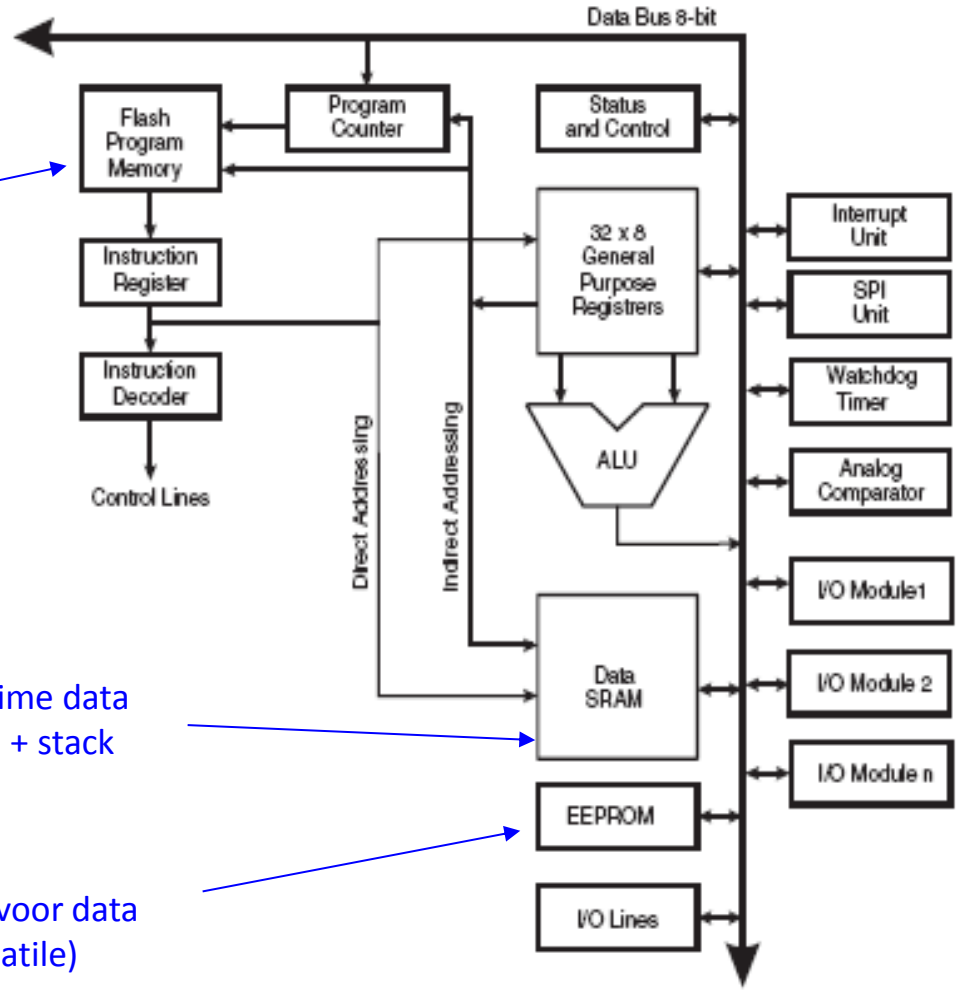


AGENDA

- intro
- de structuur van AVR assembly
- AVR instructies
- AVR registers en I/O
- AVR registers
- I/O poorten
- **ATmega memory map**
- ATMEL studio

OPBOUW AVR MCU

Figure 3. Block Diagram of the AVR MCU Architecture

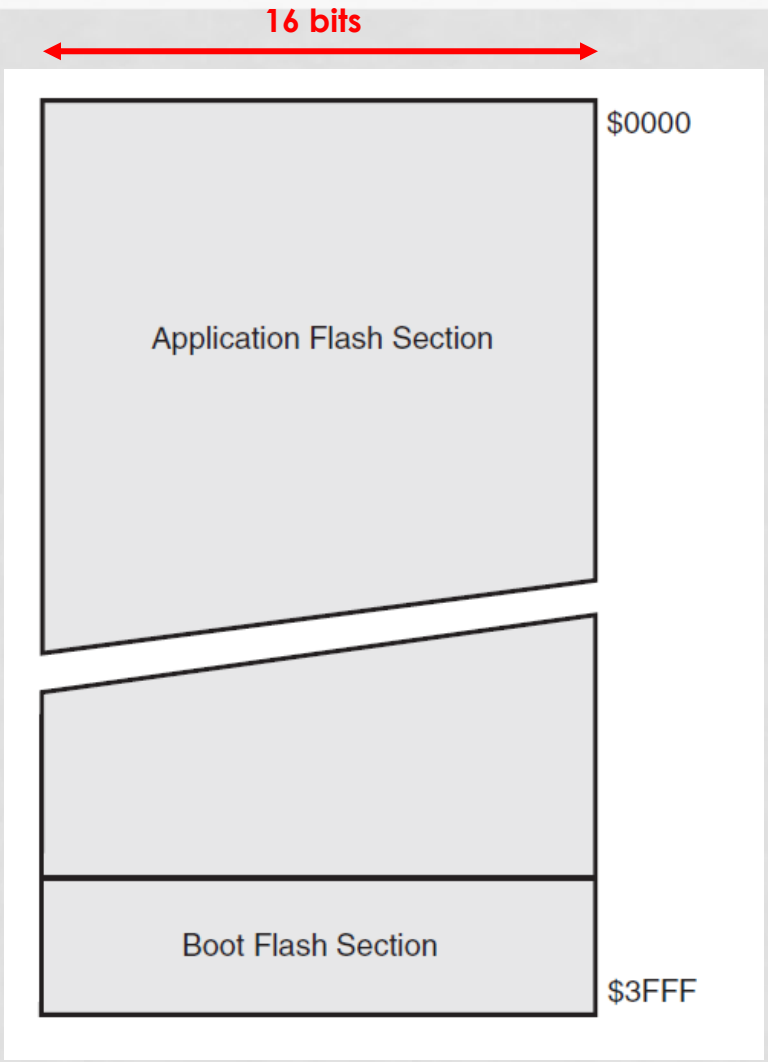


32k byte voor code
(non-volatile)

2k byte voor run-time data
globale variabelen + stack
(volatile)

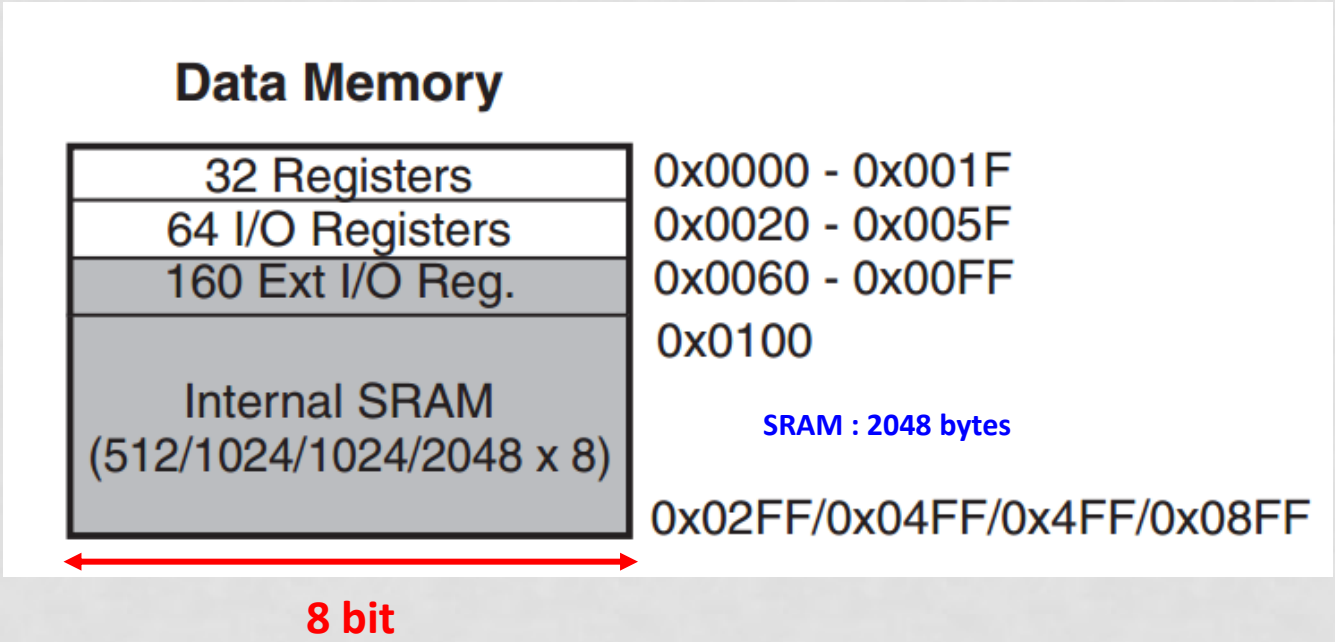
1k byte voor data
(non-volatile)

CODE MEMORY MAP (FLASH)



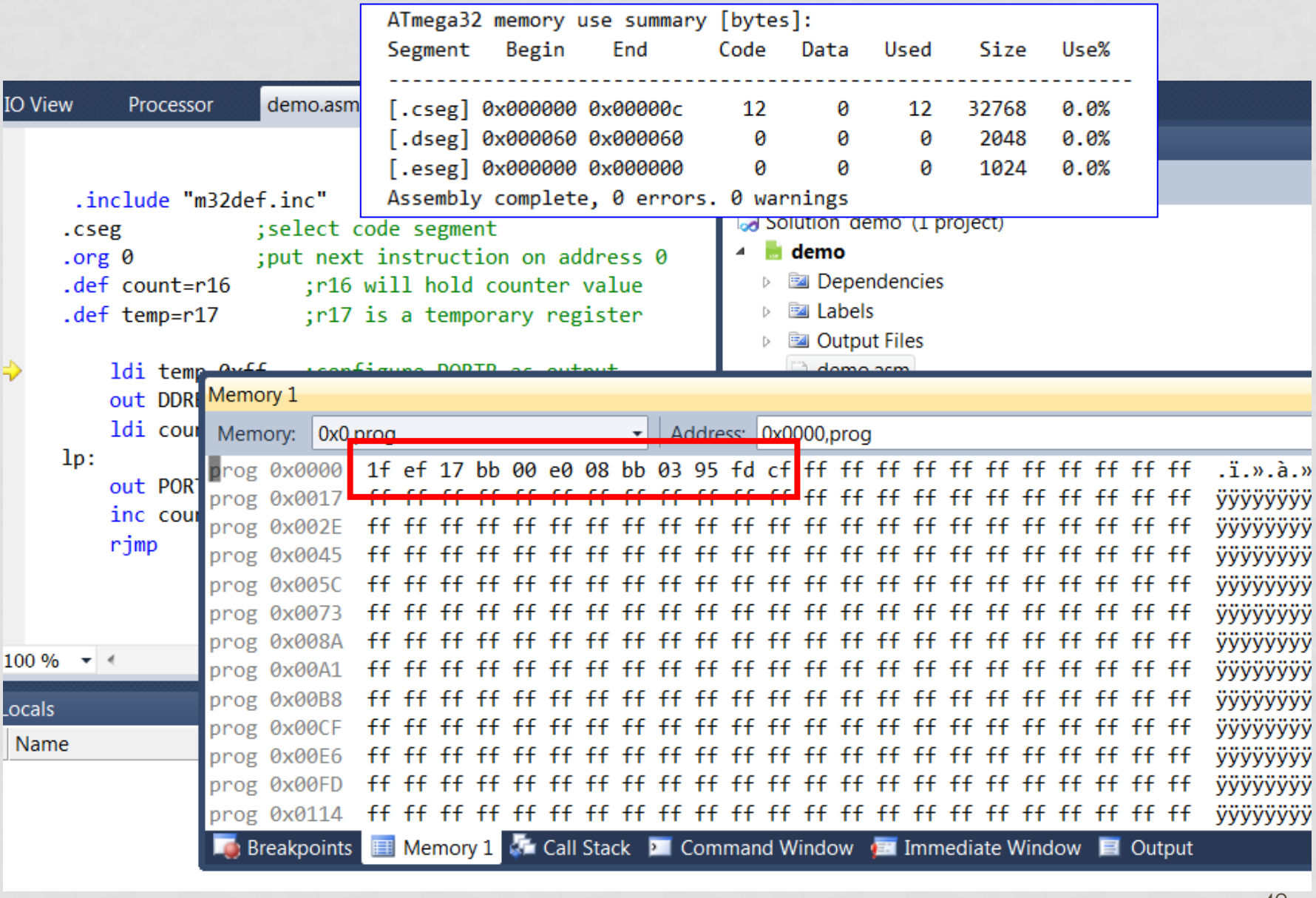
$3fff = 14 bit

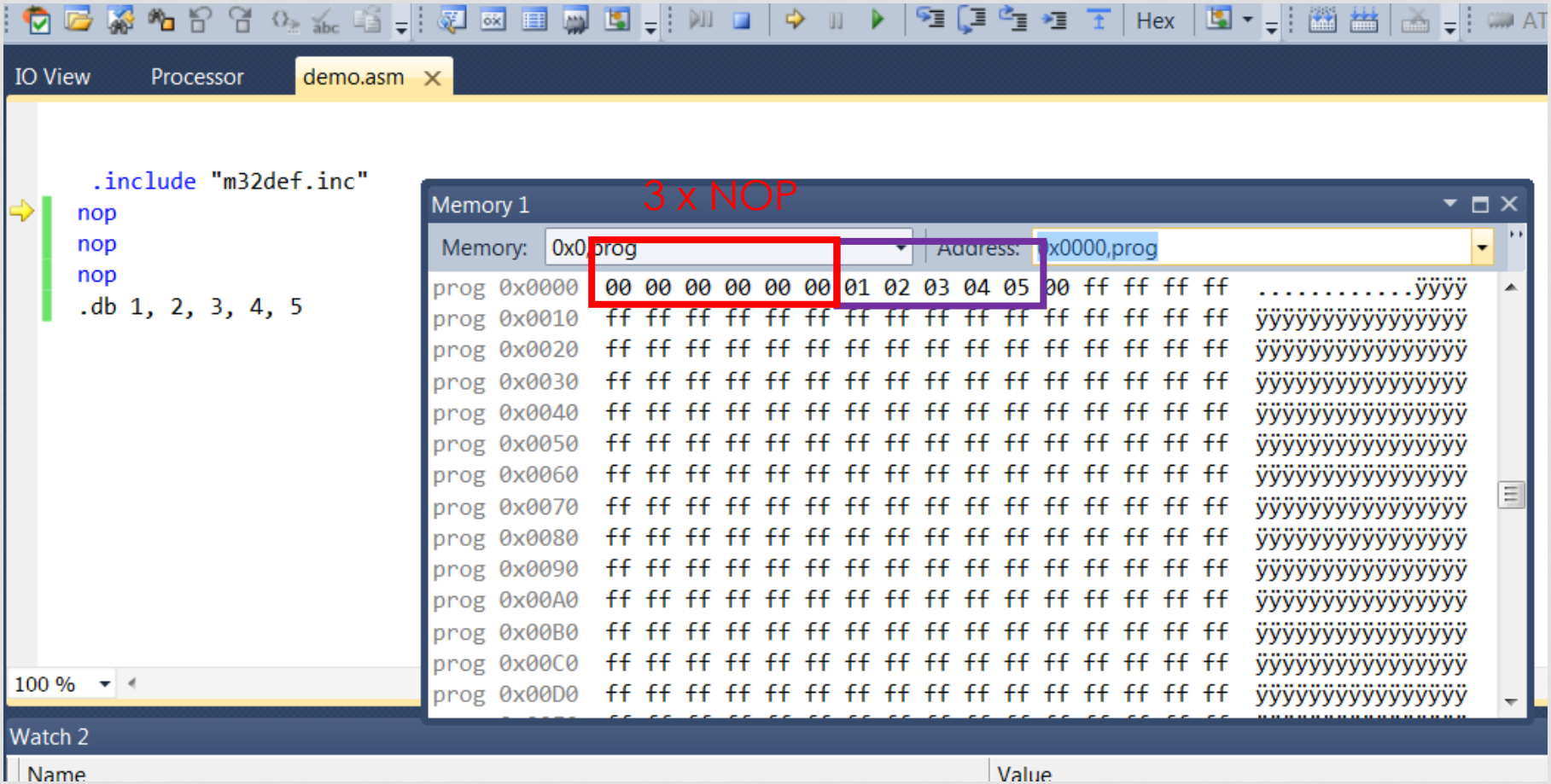
DATA MEMORY MAP (GP+I/O+SRAM)



all ATmega32 I/O and peripherals are placed in the I/O space

- 0x0020 - 0x005F : I/O space with IN/OUT instructions
- 0x060 - 0x00FF : extended I/O space with ST/STS/STD and LD/LDS/LDD instructions


ATmega32 memory use summary [bytes]:
Segment Begin End Code Data Used Size Use%
[.cseg] 0x000000 0x00000c 12 0 12 32768 0.0%
[.dseg] 0x000060 0x000060 0 0 0 2048 0.0%
[.eseg] 0x000000 0x000000 0 0 0 1024 0.0%
Assembly complete, 0 errors. 0 warnings
IO View
Processor
demo.asm
.include "m32def.inc"
.cseg ;select code segment
.org 0 ;put next instruction on address 0
.def count=r16 ;r16 will hold counter value
.def temp=r17 ;r17 is a temporary register
ldi temp,0xff ;configure PORTB as output
out DDRB,temp
ldi count,0
lp:
out PORTB,count
inc count
rjmp lp
100 %
Locals
Name
Solution 'demo' (1 project)
demo
Dependencies
Labels
Output Files
demo.asm
Memory 1
Memory: 0x0 prog
Address: 0x0000,prog
prog 0x0000 1f ef 17 bb 00 e0 08 bb 03 95 fd cf ff ff ff ff ff ff ff ff ff ff .ï.».à.»
prog 0x0017 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ÿÿÿÿÿÿÿÿ
prog 0x002E ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ÿÿÿÿÿÿÿÿ
prog 0x0045 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ÿÿÿÿÿÿÿÿ
prog 0x005C ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ÿÿÿÿÿÿÿÿ
prog 0x0073 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ÿÿÿÿÿÿÿÿ
prog 0x008A ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ÿÿÿÿÿÿÿÿ
prog 0x00A1 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ÿÿÿÿÿÿÿÿ
prog 0x00B8 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ÿÿÿÿÿÿÿÿ
prog 0x00CF ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ÿÿÿÿÿÿÿÿ
prog 0x00E6 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ÿÿÿÿÿÿÿÿ
prog 0x00FD ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ÿÿÿÿÿÿÿÿ
prog 0x0114 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ÿÿÿÿÿÿÿÿ
Breakpoints
Memory 1
Call Stack
Command Window
Immediate Window
Output



opcode NOP = 0x0000

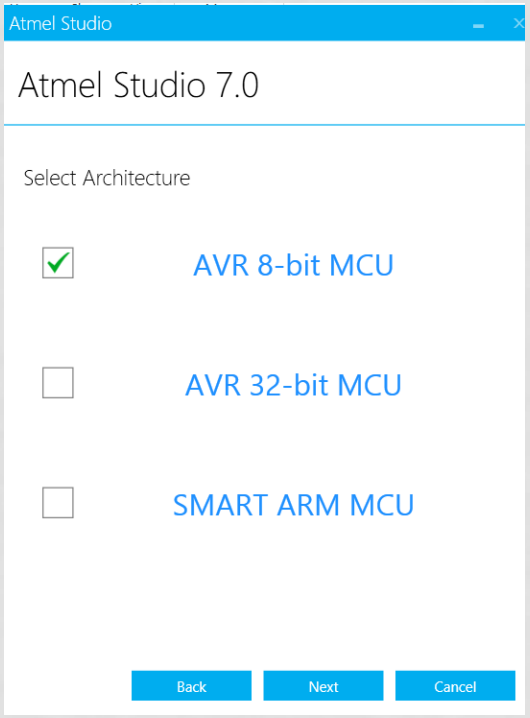
AGENDA

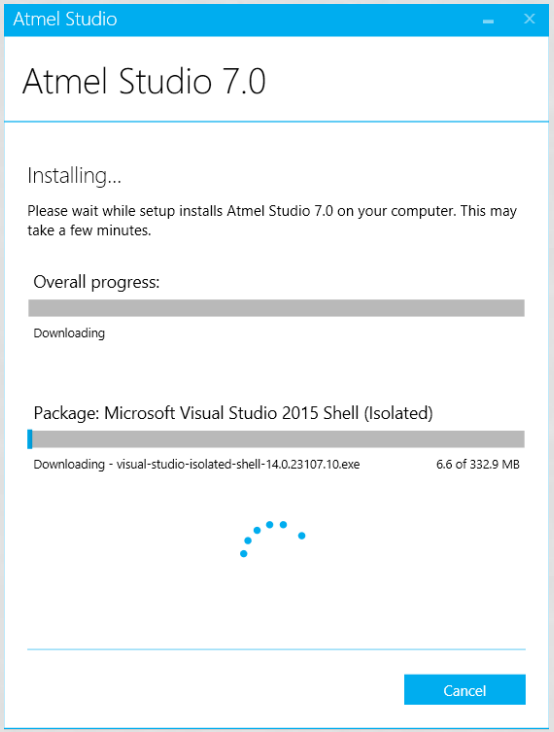
- intro
- de structuur van AVR assembly
- AVR instructies
- AVR registers
- I/O poorten
- ATMega memory map
- **ATMEL studio**

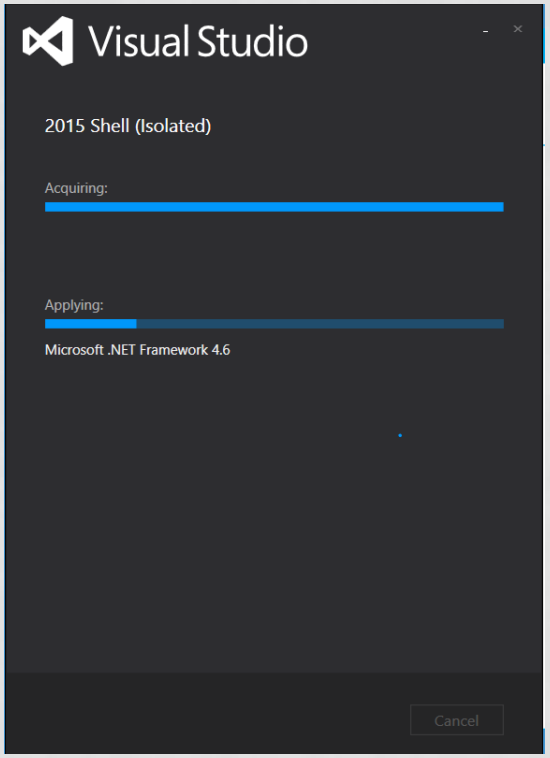
ATMEL STUDIO 6

tip : gebruik versie 6, installer staat op BB

as6installer-6.0.1996-net.exe	6-1-2013 12:15	Application	818.658 KB


Atmel Studio
Atmel Studio 7.0
Select Architecture
AVR 8-bit MCU
AVR 32-bit MCU
SMART ARM MCU
Back
Next
Cancel


Atmel Studio
Atmel Studio 7.0
Installing...
Please wait while setup installs Atmel Studio 7.0 on your computer. This may take a few minutes.
Overall progress:
Downloading
Package: Microsoft Visual Studio 2015 Shell (Isolated)
Downloading - visual-studio-isolated-shell-14.0.23107.10.exe
6.6 of 332.9 MB
Cancel

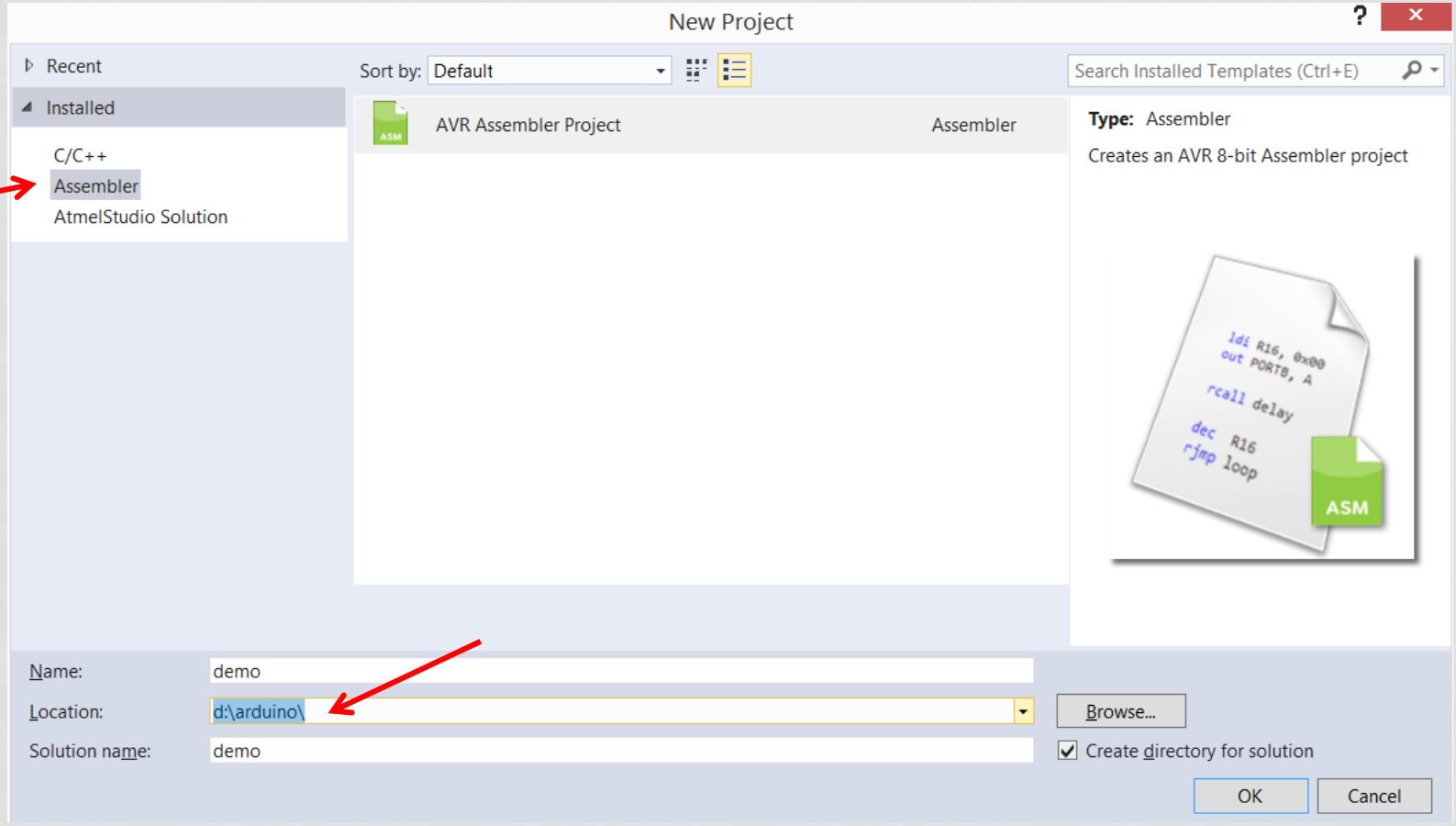

Visual Studio
2015 Shell (Isolated)
Acquiring:
Applying:
Microsoft .NET Framework 4.6
Cancel


Windows Security
Would you like to install this device software?
Name: Jungo Connectivity
Publisher: Atmel Norway
Always trust software from "Atmel Norway".
Install
Don't Install
You should only install driver software from publishers you trust. How can I decide which device software is safe to install?

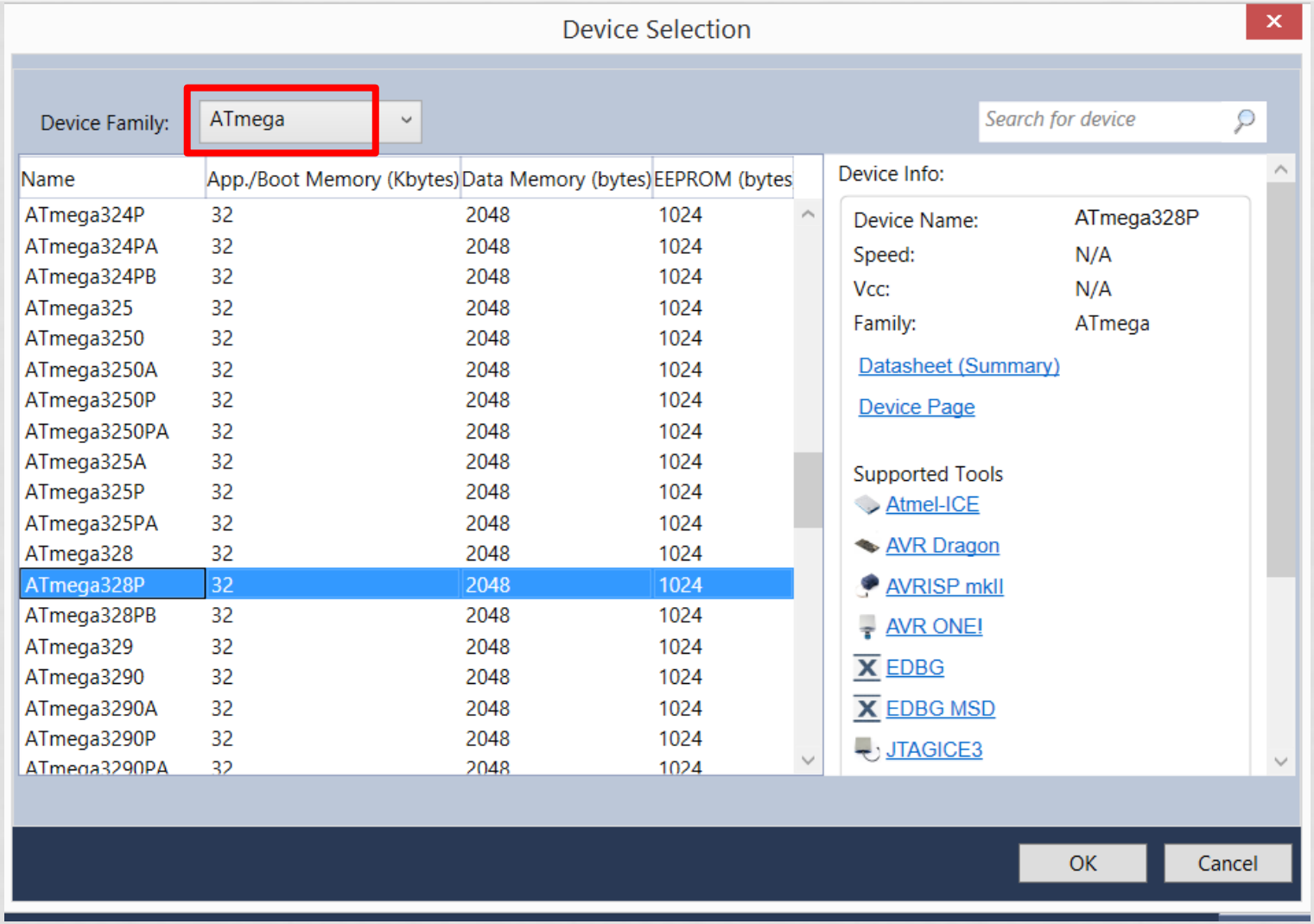
ATMEL STUDIO

- voor AVR en ARM
- een IDE op Windows
  - text editor
  - AVR assembler
  - AVR gcc (en avr-libc)
  - AVR simulator en debugger
  - programming support via serial interface
- download via www.atmel.com/tools/atmelstudio
  - installs MS .NET framework and Visual Studio Shell
  - requirements : 2 GB RAM, 2 GB hard disk space, modern video card + DirectX 9
  - may take 20+ mins !

CREATE A PROJECT



CREATE A PROJECT



BUILD A PROJECT

demo demo.asm

```
 *  Created: 14-3-2016 21:42:15
 *   Author: jacob
 */

.include "m328Pdef.inc"
.def a = r16
.def b = r17
.def c = r18

start:
  ldi a, 10
  ldi b, 11
  mov c, a
  add c, b

loop:
  rjmp loop
```

100 %

Output

Show output from: Build

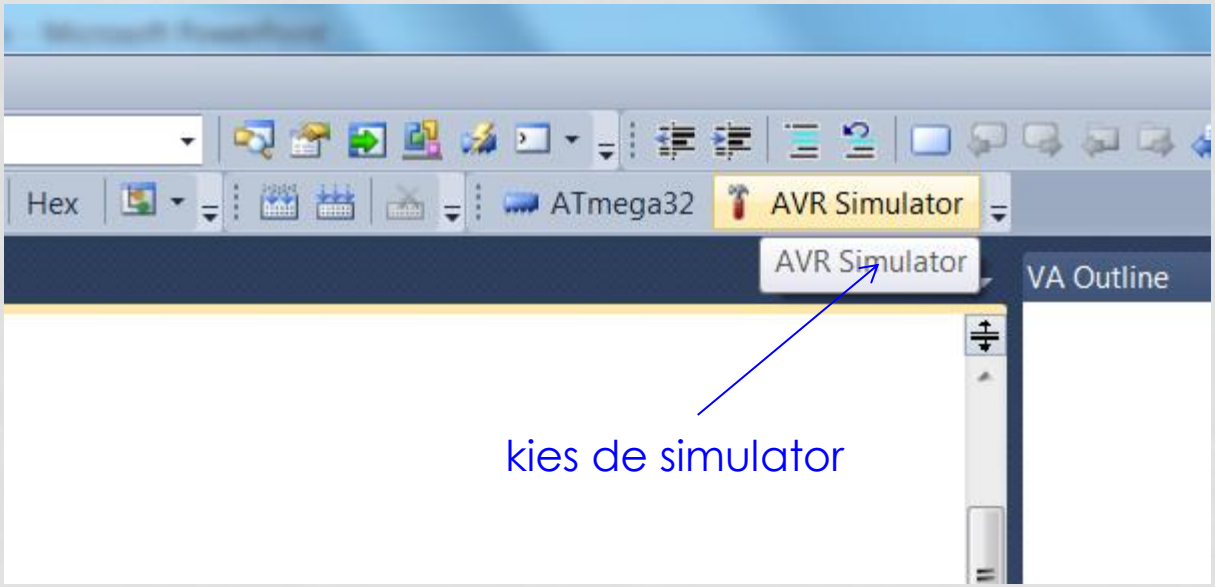
```
Target "Build" in file "D:\Program Files (x86)\Atmel\Atmel Studio 6.0\Vs\Avr
Done building target "Build" in project "demo.asmproj".
Done building project "demo.asmproj".

Build succeeded.
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

Share View

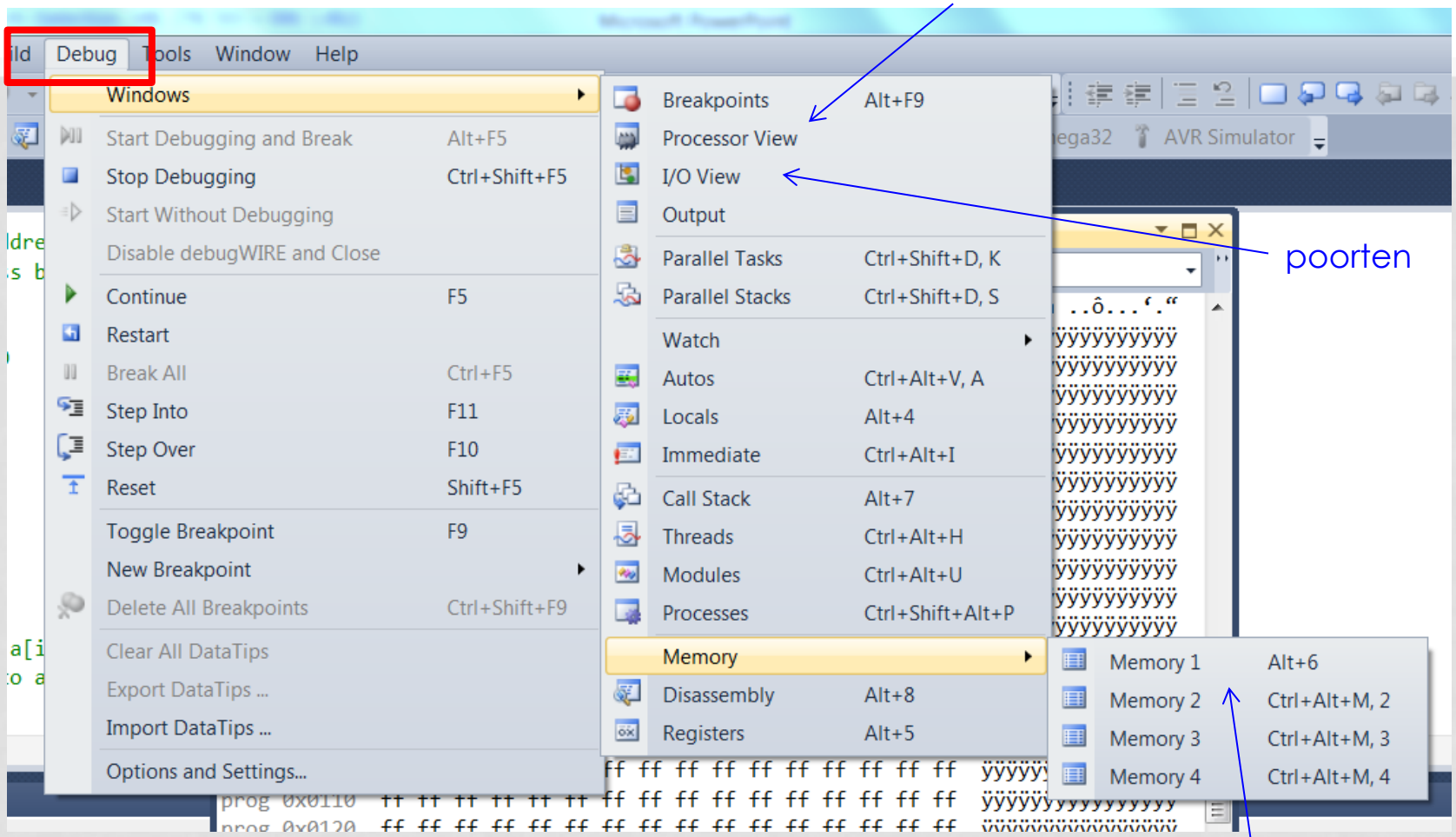
This PC ▸ DATA (D:) ▸ arduino ▸ demo ▸ demo ▸ Debug

Name	Date modified	Type	Size
demo.hex	14-3-2016 17:18	HEX File	1 KB
demo.lss	14-3-2016 17:18	LSS File	54 KB
demo.map	14-3-2016 17:18	MAP File	20 KB
demo.obj	14-3-2016 17:18	OBJ File	1 KB
demo.tmp	14-3-2016 17:18	TMP File	1 KB

DEBUGGER

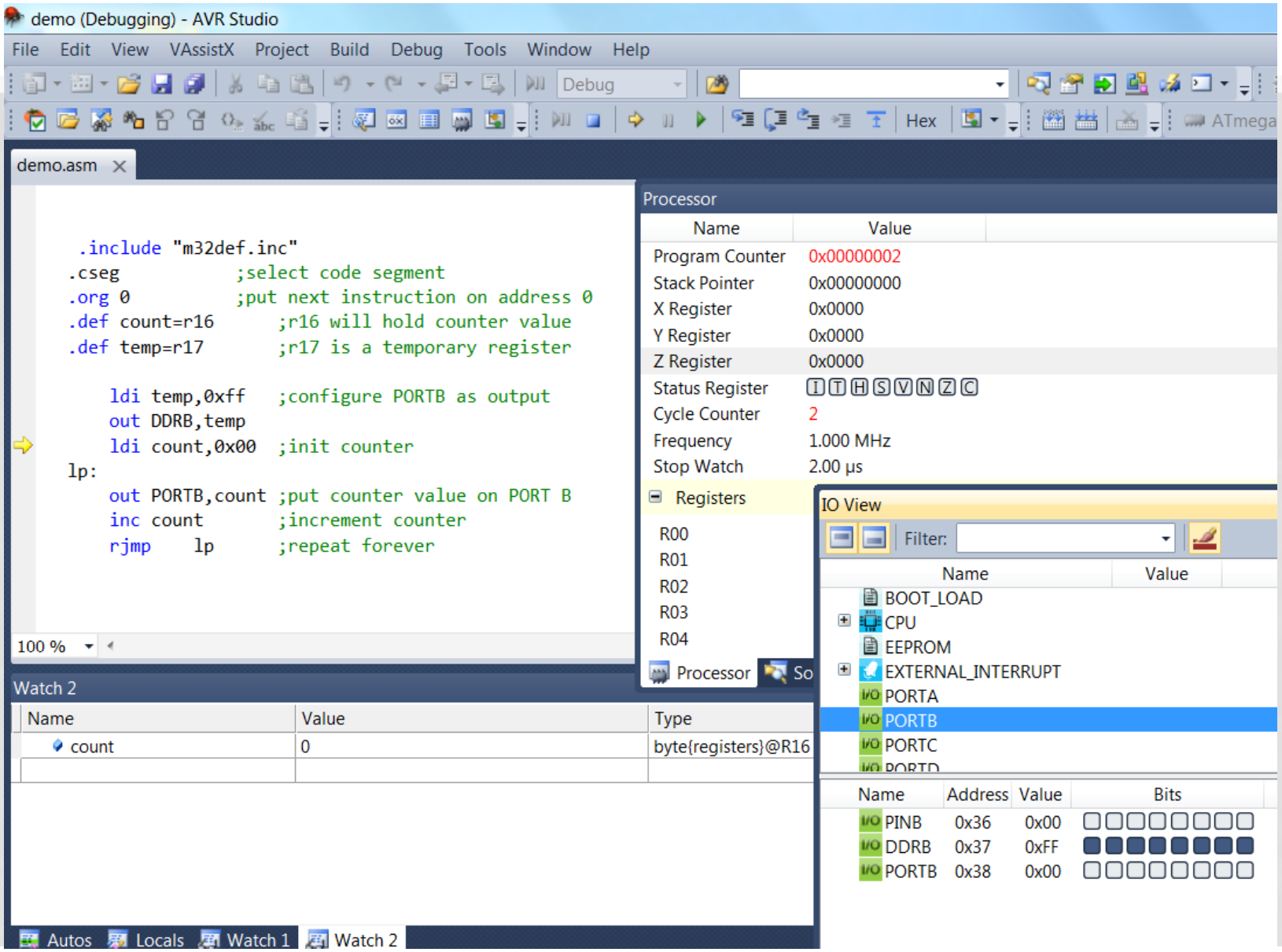

Hex
ATmega32
AVR Simulator
AVR Simulator
VA Outline
kies de simulator

DEBUGGER

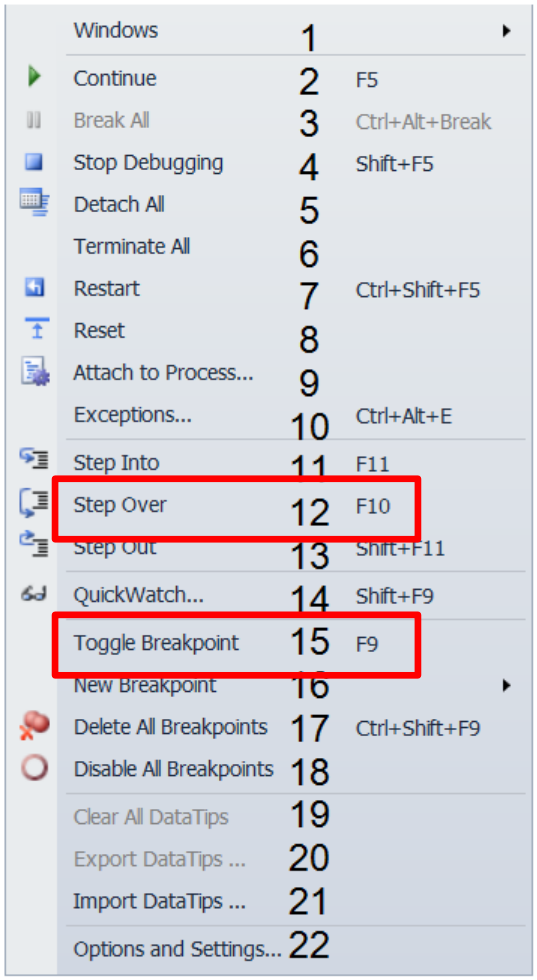


tip : in processor, I/O en memory window kun je ook waarden aanpassen

DEBUGGER


demo (Debugging) - AVR Studio
File Edit View VAssistX Project Build Debug Tools Window Help
Debug
Hex
ATmega
demo.asm
.include "m32def.inc"
.cseg ;select code segment
.org 0 ;put next instruction on address 0
.def count=r16 ;r16 will hold counter value
.def temp=r17 ;r17 is a temporary register
ldi temp,0xff ;configure PORTB as output
out DDRB,temp
ldi count,0x00 ;init counter
lp:
out PORTB,count ;put counter value on PORT B
inc count ;increment counter
rjmp lp ;repeat forever
100 %
Processor
Name Value
Program Counter 0x00000002
Stack Pointer 0x00000000
X Register 0x0000
Y Register 0x0000
Z Register 0x0000
Status Register I T H S V N Z C
Cycle Counter 2
Frequency 1.000 MHz
Stop Watch 2.00 µs
Registers
R00
R01
R02
R03
R04
Processor
IO View
Filter:
Name Value
BOOT_LOAD
CPU
EEPROM
EXTERNAL_INTERRUPT
PORTA
PORTB
PORTC
Name Address Value Bits
PINB 0x36 0x00
DDRB 0x37 0xFF
PORTB 0x38 0x00
Watch 2
Name Value Type
count 0 byte{registers}@R16
Autos Locals Watch 1 Watch 2

Debug control



1	This command shows the available tool windows
2	Continue running the active tool from the breakpoint
3	Suspend all debugging processes
4	Close the debugging mode and return to development mode
5	Detach all external debugging processes (MS Visual Studio for ex.) and threads from the solution
6	Terminate all attached debugging processes
7	Return to the initial tool state and restart debugging
8	Reset the tool state without interrupting the debugged solution
9	The Attach command is used to attach to a running application. See AVR ONE! online help for further details.
10	Determine which exceptions should be raised, suppressed, handled or ignored by which process.
11	The Step Into command in the Debug menu executes one instruction. When AVR Studio is in source mode, one source level instruction is executed, and when in disassembly level, one assembly level instruction is executed. After the Step Into is completed, all information in all windows are updated.
12	The Step Over command in the Debug menu executes one instruction. If the instruction contains a function call/subroutine call, the function/subroutine is executed as well. If a user breakpoint is encountered during Step Over, execution is halted. After the Step Over is completed, all information in all windows are updated.
13	The Step Out command in the Debug menu executes until the current function has completed. If a user breakpoint is encountered during Step Over, execution is halted. If a Step Out command is issued when the program is on the top level, the program will continue executing until it reaches a breakpoint or it is stopped by the user. After the Step Out command is completed, all information in all windows are updated.
14	Add a symbol defined as the expression under cursor
15	The Toggle breakpoint command toggles the breakpoint status for the instruction where the cursor is placed. Note that this function is only available when the source window or disassembly window is the active view.
16	Create a new breakpoint see more at Breakpoints
17	This function clears all set program breakpoints, including breakpoints which have been disabled.