# Atmega328PB Instruction Set and Programming

- Understanding ATmega328PB Instruction Set
- How to make a program using assembly language



#### ATmega328PB Instruction Set (1)

- The Assembler is not case sensitive.
- The operands have the following forms:

```
Rd: R0-R31 or R16-R31 (depending on instruction)
```

```
Rr: RØ-R31
```

- *b*: Constant (0-7), can be a constant expression
- s: Constant (0-7), can be a constant expression
- P: Constant (0-31/63), can be a constant expression
- K: Constant (0-255), can be a constant expression
- k: Constant, value range depending on instruction. Can be a constant expression.
- q: Constant (0-63), can be a constant expression

#### ATmega328PB Instruction Set (2)

Mnemonics	Operands	Description	Operation	Flags	No. of Clock
ARITHMETIC	AND LOGI	C INSTRUCTIONS			
ADD	Rd, Rr	Add without Carry	$Rd \leftarrow Rd + Rr$	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry	$Rd \leftarrow Rd + Rr + C$	Z,C,N,V,H	1
ADIW	Rd, K	Add Immediate to Word	$Rd+1:Rd \leftarrow Rd+1:Rd + K$	Z,C,N,V	2
SUB	Rd, Rr	Subtract without Carry	$Rd \leftarrow Rd - Rr$	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Immediate	$Rd \leftarrow Rd - K$	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry	$Rd \leftarrow Rd - Rr - C$	Z,C,N,V,H	1
SBCI	Rd, K	Subtract Immediate with Carry	$Rd \leftarrow Rd - K - C$	Z,C,N,V,H	1
SBIW	Rd, K	Subtract Immediate from Word	$Rd+1:Rd \leftarrow Rd+1:Rd - K$	Z,C,N,V	2
AND	Rd, Rr	Logical AND	$Rd \leftarrow Rd \bullet Rr$	Z,N,V	1
ANDI	Rd, K	Logical AND with Immediate	$Rd \leftarrow Rd \bullet K$	Z,N,V	1
OR	Rd, Rr	Logical OR	$Rd \leftarrow Rd v Rr$	Z,N,V	1
ORI	Rd, K	Logical OR with Immediate	$Rd \leftarrow Rd \vee K$	Z,N,V	1

#### ATmega328PB Instruction Set (3)

Mnemonics	Operands	Description	Operation	Flags	No. of Clock	
ARITHMETIC	ARITHMETIC AND LOGIC INSTRUCTIONS					
EOR	Rd, Rr	Exclusive OR	$Rd \leftarrow Rd \oplus Rr$	Z,N,V	1	
СОМ	Rd	One's Complement	Rd ← \$FF – Rd	Z,C,N,V	1	
NEG	Rd	Two's Complement	Rd ← \$00 – Rd	Z,C,N,V,H	1	
SBR	Rd,K	Set Bit(s) in Register	$Rd \leftarrow Rd v K$	Z,N,V	1	
CBR	Rd,K	Clear Bit(s) in Register	$Rd \leftarrow Rd \bullet (\$FFh - K)$	Z,N,V	1	
INC	Rd	Increment	$Rd \leftarrow Rd + 1$	Z,N,V	1	
DEC	Rd	Decrement	$Rd \leftarrow Rd - 1$	Z,N,V	1	
TST	Rd	Test for Zero or Minus	$Rd \leftarrow Rd \bullet Rd$	Z,N,V	1	
CLR	Rd	Clear Register	$Rd \leftarrow Rd \oplus Rd$	Z,N,V	1	
SER	Rd	Set Register	Rd ← \$FF	None	1	
MUL	Rd,Rr	Multiply Unsigned	R1, R0 $\leftarrow$ Rd $\times$ Rr	С	2	

#### ATmega328PB Instruction Set (4)

Mnemonics	Operands	Description	Operation	Flags	No. of Clock
<b>BRANCH INST</b>	RUCTIONS				
RJMP	k	Relative Jump	$PC \leftarrow PC + k + 1$	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
JMP	k	Jump	$PC \leftarrow k$	None	3
RCALL	k	Relative Call Subroutine	$PC \leftarrow PC + k + 1$	None	3
ICALL		Indirect Call to (Z)	$PC \leftarrow Z$	None	3
CALL	k	Call Subroutine	$PC \leftarrow k$	None	4
RET		Subroutine Return	PC ← STACK	None	4
RETI		Interrupt Return	PC ← STACK	1	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC $\leftarrow$ PC + 2 or 3	None	1/2/3
СР	Rd,Rr	Compare	Rd - Rr	Z,C,N,V,H	1
СРС	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H	1
CPI	Rd,K	Compare with Immediate	Rd - K	Z,C,N,V,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC $\leftarrow$ PC + 2 or 3	None	1/2/3
SBRS	Rr, b	Skip if Bit in Register Set	if (Rr(b)=1) PC $\leftarrow$ PC + 2 or 3	None	1/2/3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if(I/O(P,b)=0) PC $\leftarrow$ PC + 2 or 3	None	1/2/3
SBIS	P, b	Skip if Bit in I/O Register Set	if(I/O(P,b)=1) PC $\leftarrow$ PC + 2 or 3	None	1/2/3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC $\leftarrow$ PC + k + 1	None	1/2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC $\leftarrow$ PC + k + 1	None	1/2

#### ATmega328PB Instruction Set (5)

Mnemonics	Operands	Description	Operation	Flags	No. of Clock
BRANCH INST	RUCTIONS			·	
BREQ	k	Branch if Equal	if (Z = 1) then PC $\leftarrow$ PC + k + 1	None	1/2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC $\leftarrow$ PC + k + 1	None	1/2
BRCS	k	Branch if Carry Set	if (C = 1) then PC $\leftarrow$ PC + k + 1	None	1/2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC $\leftarrow$ PC + k + 1	None	1/2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC $\leftarrow$ PC + k + 1	None	1/2
BRLO	k	Branch if Lower	if (C = 1) then PC $\leftarrow$ PC + k + 1	None	1/2
BRMI	k	Branch if Minus	if (N = 1) then PC $\leftarrow$ PC + k + 1	None	1/2
BRPL	k	Branch if Plus	if (N = 0) then PC $\leftarrow$ PC + k + 1	None	1/2
BRGE	k	Branch if Greater or Equal, Signed	if (N $\oplus$ V= 0) then PC $\leftarrow$ PC+ k + 1	None	1/2
BRLT	k	Branch if Less Than, Signed	if (N $\oplus$ V= 1) then PC $\leftarrow$ PC + k + 1	None	1/2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC $\leftarrow$ PC + k + 1	None	1/2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC $\leftarrow$ PC + k + 1	None	1/2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC $\leftarrow$ PC + k + 1	None	1/2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC $\leftarrow$ PC + k + 1	None	1/2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC $\leftarrow$ PC + k + 1	None	1/2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC $\leftarrow$ PC + k + 1	None	1/2
BRIE	k	Branch if Interrupt Enabled	if $(I = 1)$ then PC $\leftarrow$ PC + k + 1	None	1/2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC $\leftarrow$ PC + k + 1	None	1/2

### ATmega328PB Instruction Set (6)

Mnemonics	Operands	Description	Operation	Flags	No. of Clock
DATA TRANSFE		ONS			
MOV	Rd, Rr	Copy Register	$Rd \leftarrow Rr$	None	1
LDI	Rd, K	Load Immediate	$Rd \leftarrow K$	None	1
LDS	Rd, k	Load Direct from SRAM	$Rd \leftarrow (k)$	None	3
LD	Rd, X	Load Indirect	$Rd \leftarrow (X)$	None	2
LD	Rd, X+	Load Indirect and Post-Increment	$Rd \leftarrow (X), X \leftarrow X + 1$	None	2
LD	Rd, -X	Load Indirect and Pre-Decrement	$X \leftarrow X - 1$ , $Rd \leftarrow (X)$	None	2
LD	Rd, Y	Load Indirect	$Rd \leftarrow (Y)$	None	2
LD	Rd, Y+	Load Indirect and Post-Increment	$Rd \leftarrow (Y), Y \leftarrow Y + 1$	None	2
LD	Rd, -Y	Load Indirect and Pre-Decrement	$Y \leftarrow Y - 1$ , $Rd \leftarrow (Y)$	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	$Rd \leftarrow (Y + q)$	None	2
LD	Rd, Z	Load Indirect	$Rd \leftarrow (Z)$	None	2
LD	Rd, Z+	Load Indirect and Post-Increment	$Rd \leftarrow (Z), Z \leftarrow Z+1$	None	2
LD	Rd, -Z	Load Indirect and Pre-Decrement	$Z \leftarrow Z - 1$ , $Rd \leftarrow (Z)$	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	$Rd \leftarrow (Z + q)$	None	2
STS	k, Rr	Store Direct to SRAM	$(k) \leftarrow Rr$	None	3
ST	X, Rr	Store Indirect	$(X) \leftarrow Rr$	None	2

### ATmega328PB Instruction Set (7)

Mnemonics	Operands	Description	Operation	Flags	No. of Clock
DATA TRANSFE		ONS CONTRACTOR OF			
ST	X+, Rr	Store Indirect and Post-Increment	$(X) \leftarrow Rr, X \leftarrow X + 1$	None	2
ST	-X <i>,</i> Rr	Store Indirect and Pre-Decrement	$X \leftarrow X - 1$ , $(X) \leftarrow Rr$	None	2
ST	Y, Rr	Store Indirect	$(Y) \leftarrow Rr$	None	2
ST	Y+, Rr	Store Indirect and Post-Increment	$(Y) \leftarrow Rr, Y \leftarrow Y + 1$	None	2
ST	-Y, Rr	Store Indirect and Pre-Decrement	$Y \leftarrow Y - 1$ , $(Y) \leftarrow Rr$	None	2
STD	Y+q,Rr	Store Indirect with Displacement	$(Y + q) \leftarrow Rr$	None	2
ST	Z, Rr	Store Indirect	$(Z) \leftarrow Rr$	None	2
ST	Z+, Rr	Store Indirect and Post-Increment	$(Z) \leftarrow Rr,  Z \leftarrow Z + 1$	None	2
ST	-Z, Rr	Store Indirect and Pre-Decrement	$Z \leftarrow Z - 1$ , (Z) $\leftarrow Rr$	None	2
STD	Z+q,Rr	Store Indirect with Displacement	$(Z + q) \leftarrow Rr$	None	2
LPM		Load Program Memory	$R0 \leftarrow (Z)$	None	3
IN	Rd, P	In Port	$Rd \leftarrow P$	None	1
OUT	P, Rr	Out Port	P←Rr	None	1
PUSH	Rr	Push Register on Stack	$STACK \leftarrow Rr$	None	2
POP	Rd	Pop Register from Stack	$Rd \leftarrow STACK$	None	2

#### ATmega328PB Instruction Set (8)

Mnemonics	Operands	Description	Operation	Flags	No. of Clock
BIT AND BIT-	TEST INSTRU	CTIONS			
LSL	Rd	Logical Shift Left	$Rd(n+1) \leftarrow Rd(n), Rd(0) \leftarrow 0, C \leftarrow Rd(7)$	Z,C,N,V,H	1
LSR	Rd	Logical Shift Right	$Rd(n) \leftarrow Rd(n+1), Rd(7) \leftarrow 0, C \leftarrow Rd(0)$	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	$Rd(0) \leftarrow C, Rd(n+1) \leftarrow Rd(n), C \leftarrow Rd(7)$	Z,C,N,V,H	1
ROR	Rd	Rotate Right Through Carry	$Rd(7) \leftarrow C, Rd(n) \leftarrow Rd(n+1), C \leftarrow Rd(0)$	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=06	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	$Rd(30) \leftrightarrow Rd(74)$	None	1
BSET	S	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	S	Flag Clear	SREG(s) ← 0	SREG(s)	1
SBI	P, b	Set Bit in I/O Register	I/O(P, b) ← 1	None	2
CBI	P, b	Clear Bit in I/O Register	I/O(P, b) ← 0	None	2
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	Т	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	С	1
CLC		Clear Carry	C ← 0	С	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← Ø	N	1

#### ATmega328PB Instruction Set (9)

Mnemonics	Operands	nds Description Operation Flags		Flags	No. of Clock
BIT AND BIT-TEST INSTRUCTIONS					
CLN		Clear Negative Flag	$N \leftarrow 0$	Ν	1
SEZ		Set Zero Flag	$Z \leftarrow 1$	Z	1
CLZ		Clear Zero Flag	Z < 0	Z	1
SEI		Global Interrupt Enable	$I \leftarrow 1$	I	1
CLI		Global Interrupt Disable	$I \leftarrow 0$	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Two's Complement Overflow	$V \leftarrow 1$	V	1
CLV		Clear Two's Complement Overflow	$V \leftarrow 0$	V	1
SET		Set T in SREG	T ← 1	Т	1
CLT		Clear T in SREG	T ← 0	Т	1
SEH		Set Half Carry Flag in SREG	$H \leftarrow 1$	Н	1
CLH		Clear Half Carry Flag in SREG	H ← 0	Н	1
NOP		No Operation		None	1
SLEEP		Sleep		None	1
WDR		Watchdog Reset		None	1

- An assembly language statement line may take one of the four following forms:
  - > [label:] directive [operands] [Comment]
  - > [label:] instruction [operands] [Comment]
  - Comment
  - Empty line
- A comment has the following form:
  - ; [Text]
- Items placed in braces are optional.
- The text between the comment-delimiter (;) and the end of line (EOL) is ignored by the Assembler.
- Examples:

•	label:	.EQU var1=100	; Set var1 to 100 (Directive)
		.EQU var2=200	; Set var2 to 200
	test: ; Pure	rjmp test comment line	; Infinite loop (Instruction)
	-		; Another comment line

Note: There are no restrictions with respect to column placement of labels, directives, comments or instructions.

- The directives are not translated directly into opcodes.
- They are used to adjust the location of the program in memory, define macros, initialize memory and so on.

# Assembler Directives (2)

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

Directive Description **ESEG EEPROM Segment** Exit from file EXIT Read source from another file INCLUDE LIST Turn list file generation on LISTMAC Turn macro expansion on MACRO Begin macro Turn list file generation off NOLIST Set program origin ORG SET Set a symbol to an expression

Note: All directives must be preceded by a period.

### Assembler Directives (3)

#### 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.

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

### Assembler Directives (4)

#### DSEG – Data Segment

The DSEG directive defines the start of a Data Segment. An Assembler file can consist of several Data Segments, which are concatenated into one Data Segment when assembled. A Data Segment will normally only consist of BYTE directives (and labels). The Data Segments have their own location counter which is a byte counter. The ORG directive (see description later in this document) can be used to place the variables at specific locations in the SRAM. The directive does not take any parameters.

	.DSEG		
	.DSEG		
var1:	.BYTE	1	; reserve 1 byte to var1
table:	.BYTE	6	; reserve 6 bytes to table
	.CSEG		
	ldi r30	,low(var1)	; Load Z register low
	ldi r31	,high(var1)	; Load Z register high
	ld r1,	Z	; Load var1 into r1

### Assembler Directives (5)

#### ESEG – EEPROM Segment

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

	.ESEG		
	.DSEG		; Start data segment
vartab:	.BYTE	4	; Reserve 4 bytes in SRAM
	.ESEG		
eevar:	.DW	0xff0f	; Initialize one word in EEPROM
	.CSEG		; Start code segment
const:	.DW	2	; Write 0x0002 in prog.mem.
	mov	r1,r0	; Do something

## Assembler Directives (6)

#### BYTE – Reserve bytes to a variable

The BYTE directive reserves memory resources in the SRAM. In order to be able to refer to the reserved location, the BYTE directive should be preceded by a label. The directive takes one parameter, which is the number of bytes to reserve. The directive can only be used within a Data Segment (see directives CSEG, DSEG and ESEG). Note that a parameter must be given. The allocated bytes are not initialized.

LABEL:	.BYTI	E express	ion
	.DSEG		
var1:	.BYTE	1	; reserve 1 byte to var1
table:	.BYTE	6	; reserve 6 bytes to table
	.CSEG		
	ldi r30	,low(var1)	; Load Z register low
	ldi r31	,high(var1)	; Load Z register high
	ld r1,Z		; Load VAR1 into register 1

## Assembler Directives (7)

*DB* – Define constant byte(s) in program memory or EEPROM memory

The DB directive reserves memory resources in the program memory or the EEPROM memory. In order to be able to refer to the reserved locations, the DB directive should be preceded by a label.

The DB directive takes a list of expressions, and must contain at least one expression. The DB directive must be placed in a Code Segment or an EEPROM Segment.

The expression list is a sequence of expressions, delimited by commas. Each expression must evaluate to a number between -128 and 255. If the expression evaluates to a negative number, the 8 bits two's complement of the number will be placed in the program memory or EEPROM memory location.

If the DB directive is used in a Code Segment and the expression\_list contains more than one expression, the expressions are packed so that two bytes are placed in each program memory word. If the expression\_list contains an odd number of expressions, the last expression will be placed in a program memory word of its own, even if the next line in the assembly code contains a DB directive.

LABEL:	.DB	expression_list
consts:	.CSEG .DB .ESEG	0, 255, 0b01010101, -128, 0xaa
eeconst:	.DB	Øxff

#### Assembler Directives (8)

<i>DW</i> – Define constant word(s) in program memory or E2PROM memory			
The DW directive reserves memory resources in the program memory or EEPROM memory. In order to be able to refer to			
the reserved locations, the DW directive should be preceded by a label.			
The DW directive takes a list of expressions, and must contain at least one expression.			
The DW directive must be placed in a Code Segment or an EEPROM Segment.			
The expression list is a sequence of expressions, delimited by commas. Each expression must evaluate to a number			
between -32768 and 65535. If the expression evaluates to a negative number, the 16 bits two's complement of the number			
will be placed in the program memory location.			
LABEL: .DW expression_list			
.CSEG			
varlist: . <i>DW</i> 0,0xffff,0b1001110001010101,-32768,65535			
.ESEG			
eevar: .DW 0xffff			

### Assembler Directives (9)

*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.

LABEL: .DEF Symbol = Register

.DEF temp = R16	
.DEF ior = RØ	
.CSEG	
ldi <mark>temp</mark> ,0xf0	; Load 0xf0 into temp register
in ior,0x3f	; Read SREG into ior register
eor temp,ior	; Exclusive or temp and ior

#### Assembler Directives (10)

#### 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.

. EQU	label = expression
. EQU	<pre>io_offset = 0x23</pre>
. EQU	porta = io_offset + 2
.CSEG	; Start code segment
clr r2	; Clear register 2
out port	ta,r2 ; Write to Port A

#### Assembler Directives (11)

#### SET – Set a symbol equal to an expression

The SET directive assigns a value to a label. This label can then be used in later expressions.

A label assigned to a value by the SET directive can be changed later in the program.

.SET	label = expression	
.SET	<pre>io_offset = 0x23</pre>	
.SET	<pre>porta = io_offset + 2</pre>	
.CSEG	; Start code segment	
clr r2	; Clear register 2	
out por	ta,r2 ; Write to Port A	

## Assembler Directives (12)

ORG – Set program origin

The ORG directive sets the location counter to an absolute value. The value to set is given as a parameter. If an ORG directive is given within a Data Segment, then it is the SRAM location counter which is set. If the directive is given within a Code Segment, then it is the Program memory counter which is set. If the directive is given within an EEPROM Segment, then it is the EEPROM location counter which is set. If the directive is preceded by a label (on the same source code line), the label will be given the value of the parameter. The default values of the Code and EEPROM location counters are zero, whereas the default value of the SRAM location counter is 32 (due to the registers occupying addresses 0-31) when the assembling is started. Note that the EEPROM and SRAM location counters count bytes whereas the Program memory location counter counts words.

	.ORG	expression	
	.DSEG	; Start data segment 0x67   ; Set SRAM address to hex 67	
rvar:	.BYTE	1 ; Reserve a byte at SRAM address 67H : Start FEPROM Segment	
eevar:	. ORG	0x20 ; Set EEPROM location counter 0xfeff : Initialize one word	
	.CSEG .ORG	0x10 ; Set Program Counter to hex 10	
	mov	ru,ri ; Do something	

#### Assembler Directives (13)

INCLUDE – Include another file			
The INCLUDE directive tells the Assembler to start reading from a specified file. The Assembler then assembles the			
specified file until end of file (EOF) or an E	EXIT directive is encountered. An included file may itself contain INCLUDE		
directives.			
.INCLUDE "filename"			
.EQU sreg = 0x3f	; Status register		
.EQU sphigh = 0x3e	; Stack pointer high		
.EQU splow = 0x3d	; Stack pointer low		
	; incdemo.asm		
.INCLUDE "iodefs.asm"	; Include I/O definitions		
in r0,sreg	; Read status register		

# Expressions (1)

Expressions can consist of operands, operators and functions. All expressions are internally 32 bits.

#### Operands

The following operands can be used:

- User defined labels which are given the value of the location counter at the place they appear.
- User defined variables defined by the SET directive
- User defined constants defined by the EQU directive
- Integer constants: constants can be given in several formats, including,
  - ➢ Decimal (default): 10, 255
  - > Hexadecimal (two notations): 0x0a, \$0a, 0xff, \$ff
  - > Binary: 0b00001010, 0b1111111
- PC the current value of the Program memory location counter

# Expressions (2)

#### **Functions**

The following functions are defined:

- LOW(expression) returns the low byte of an expression
- HIGH(expression) returns the second byte of an expression
- BYTE2(expression) is the same function as HIGH
- BYTE3(expression) returns the third byte of an expression
- BYTE4(expression) returns the fourth byte of an expression
- LWRD(expression) returns bits 0-15 of an expression
- HWRD(expression) returns bits 16-31 of an expression
- PAGE(expression) returns bits 16-21 of an expression
- EXP2(expression) returns 2^expression
- LOG2(expression) returns the integer part of log2(expression)

# Expressions (3): Operators

Logical NOT	
Symbol: !	
Description: Unary operator which r	eturns 1 if the expression was zero, and returns 0 if the
expression was nonzero	
Precedence: 14	
Example: Ldi r16,!0xf0	; Load r16 with 0x00

#### **Bitwise NOT**

Symbol: ~	
Description: Unary operator which retu	rns the input expression with all bits inverted
Precedence: 14	
Example: Ldi r16,~0xf0	; Load r16 with 0x0f

## Expressions (4): Operators

Un	ary Minus		
	Symbol: -	: Unary operator wh	nich returns the arithmetic negation of an expression
	Precedence	e: 14	
	Example:	ldi r16,-2	; Load -2(0xfe) in r16

#### Multiplication

Symbol: *				
Description: Binary operator which returns the product of two expressions				
Precedence: 13				
Example:	ldi r30,label*2	; Load r30 with label*2		

# Expressions (5): Operators

Di	ivision		
	Symbo	ol: /	
	Descri	ption: Unary operator which r	eturns the arithmetic negation of an expression
	Preced	lence: 13	
	Examp	ole: Ldi r30,Label/2	; Load r30 with label/2

#### Addition

Symbol: +			
Description: Binary operator which returns the sum of two expressions			
Precedence: 12			
Example: <i>Ldi r30,c1+c2</i>	; Load r30 with c1+c2		

# **Expressions (6): Operators**

S	ubtraction	
	Symbol: -	
	Description: Binary operator which	returns the left expression minus the right expression
	Precedence: 12	
	Example: <i>Ldi r17,c1-c2</i>	; Load r17 with c1-c2

#### Shift left

Symbol: <<				
Description: Binary operator which returns the left expression shifted left a number of times given by				
the right expression				
Precedence: 11				
Example: Ldi r17,1<<5	; Load r17 with 1 shifted left 5 times			

# Expressions (7): Operators

Shi	ft right	
	Symbol: >> Description: Binary operator which return by the right expression	ns the left expression shifted right a number of times given
	Precedence: 11	
	Example: <i>Ldi r17,c1&gt;&gt;c2</i>	; Load r17 with c1 shifted right c2 times

#### **Bitwise AND**

Symbol: &			
Description: Binary operator which returns the bitwise And between two expressions			
Precedence: 8			
Example:	ldi r18,High(c1&c2)	;	Load r18 with an expression

# **Expressions (8): Operators**

Bi	twise XOR		
	Symbol: ^		
	Descriptio	n: Binary operator which returns	the bitwise Exclusive Or between two expressions
	Precedenc	e: 7	
	Example:	ldi r18,Low(c1^c2)	; Load r18 with an expression

#### **Bitwise OR**

Symbol:				
Description: Binary operator which returns the bitwise Or between two expressions				
Precedence: 6				
Example: Ldi r18,Low(c1/c2)	; Load r18 with an expression			

- Define names for registers with the .DEF directive, never use them with their direct name Rx.
- If you need pointer access reserve R26 to R31 for that purpose.
- A 16-bit-counter is best located in R25:R24.
- If you need to read from the program memory reserve Z (R31:R30) and R0 for that purpose.
- If you need to access to single bits within certain registers, use R16 to R23 for that purpose.
- Registers necessary for math are best placed to R1 to R15.
- If you have more than enough registers available, place all your variables in registers.
- If you get short in registers, place as many variables as necessary to SRAM.

- Define names for registers with the .DEF directive, never use them with their direct name Rx.
- If you need pointer access reserve R26 to R31 for that purpose.
- A 16-bit-counter is best located in R25:R24.
- If you need to read from the program memory reserve Z (R31:R30) and R0 for that purpose.
- If you need to access to single bits within certain registers, use R16 to R23 for that purpose.
- Registers necessary for math are best placed to R1 to R15.
- If you have more than enough registers available, place all your variables in registers.
- If you get short in registers, place as many variables as necessary to SRAM.
# Mixed Language Programming (1)

Example: A C language function calls an assembly language function.

```
/* asm_files.s */
/* main.c */
#include <avr/io.h>
                                             #define SFR OFFSET 0
                                             #include <avr/io.h>
extern void asm func(uint8 t val);
                                             .global asm func
int main(void)
                                             .section .text
{
    DDRB = 0xFF;
                                             asm func:
    asm func(3);
                                                          PORTB,R24
                                                     OUT
}
                                                     RET
```

*#define \_\_\_SFR\_OFFSET 0* To use these addresses in in/out instructions, you must subtract 0x20 from them.

# Mixed Language Programming (2)

#### Passing Parameters between C and Assembly Functions

- Parameters are passed via R25:R8 (R25 to R8).
- Parameters are passed left to right.

#### uint8\_t function(uint8\_t i, uint8\_t j);

- In this example,
  - i would be stored in R25:24 (with the actual 8-bit value stored in R24).
  - $\succ$  *j* would be stored in R23:22 (with the actual 8-bit value stored in R22).
- If the parameters passed require more memory than is available in the registers R25:R8, then the stack is used to pass additional parameters.
- Return values are placed in registers beginning at R25.
  - An 8-bit value gets returned in R24.
  - An 16-bit value gets returned in R25:R24.
  - ➢ An 32-bit value gets returned in R25:R22.
  - ➢ An 64-bit value gets returned in R25:R18.

- Atmel Studio Software
  - ✓ Integrated Development Environment (IDE)
  - ✓ Edit, compile, link, debug
  - ✓ Simulation and FLASH programming
  - ✓ Includes GNU C/C++ compiler

### Application Development Tools (2)

- ATmega328P(B) Xplained Mini Evaluation Kit Hardware
  - ✓ On-board debugger with full source-level debugging support in Atmel Studio.
  - Auto-ID for board identification in Atmel Studio 7
  - ✓ Access to all signals on target MCU.
  - ✓ One yellow user LED / One mechanical user pushbutton
  - ✓ Virtual COM port (CDC)
  - ✓ 16MHz target clock
  - ✓ USB powered
  - ✓ Arduino shield compatible foot prints
  - Supported with application examples published on Atmel Spaces



### Download and Install Atmel Studio 7

### http://www.microchip.com/development-tools/atmel-studio-7

Downloads Documentation			
Atmel Studio 7			
Title	Date Published	Size	D/L
Windows (x86/x64)			
Atmel Studio 7.0 (build 1417) web installer (recommended) - This installer contains Atmel Studio 7.0 with Atmel Software Framework 3.32.0 and Atmel Toolchains. It is recommended to use this installer if you have internet access while installing.	March 2017	2.39 MB	<b>1</b> 2
Atmel Studio 7.0 (build 1417) offline installer - This installer contains Atmel Studio 7.0 with Atmel Software Framework 3.32.0 and Atmel Toolchains. Use this installer if you do not have internet access while installing. SHA1: be58191d4f96c8333c8016ad48160132a05b2b2a	March 2017	886 MB	ŧ <u>د</u> ]
Release Notes			
Atmel Studio 7.0 Release Notes	March 2017	894 KB	5
Biomedical Engineering, Inje University			

### ATmega328P(B) Xplained Mini Evaluation Kit



# Create an AVR Application in Assembly Language (1)

- 1. Run AVR Studio 7.
- Connect ATmega328P(B) Xplained
   Mini to PC via USB Micro-B cable.
- Image of the detected ATmega328P
   Xplained Mini is displayed with related information.

ATmega328P Xplained Mi	ni - 0528 - AtmelStudio (Administrator)	▼ Quick Launch (Ctrl+Q) 🛛 🗭 🗖
File Edit View VAssistX	ASF Project Debug Tools Window Help	
G - O 👌 - 🕘 🛅 - 1	盐 🔐 🕺 🖧 🗇 🏦 📁 🗢 🤍 📄 💷 🔍 🕨 👘 🗸 Debug Brov	rser 👻 🚽 🎜 💭 🖓 🙄
∭ È ∎ → II ▶ &	* ? * ト エ Hex 경 📓 - 🚅 唱 📾 🖨 📓 🔹 🖄 💭 🚅 (	🗰 No Device 🥤 No Tool 🖕
ATmega328P Xplained Mini - 05	528 ↔ × Start Page	✓ Solution Explorer ✓ ₽
MCU board	ATmega328P Xplained Mini	▲ ○○☆   ۶
ATmega328P Xplained Mini	Armeguszor Apianea Mini	
Extension		
	The Atmel® ATmega328P Xplained Mini evalutation kit is a hardware platform to evaluate the Atmel AVR microcontrollers. The evaluation kit comes with a tubly integrated debugger that provides seamless integration with Atmel Studio 6.2 (and later revisions). The kit provides access to the features of the Atmega328P enabling easy integration of the device in a custom design.	
	Launch Data Visualizer	
	Update Kit Information	
	Atmel Studio Help:	
	Kit Userguide	
	External Links:	
	Xplained Mini Home Page	
	Xplained Mini on Spaces	
	Technical Documentation	
	ATmega328P Device Datasheet	
	Kit Details	
	Serial number ATML2323040200010528	
	Board name ATmega328P Xplained Mini	
Show page on connect	Manufacturer Atmel Norway AS	VA View Solution Explorer
Show output from:	_	• #
snow output nom.		
Error List Output		

### Create an AVR Application in Assembly Language (2)

1. Select

File – New – Project...



### Create an AVR Application in Assembly Language (3)

New Project			8 ×
₽ Recent	Sort by: Default	• # E	Search Installed Templates (Ctrl+E)
▲ Installed	(2) AVR Assembler Pro	ject Assembler	Type: Assembler
C/C++ Assembler AtmelStudio Solutio	n	2	Creates an AVR 8-bit Assembler project
Your project name			rjmp 100p ASM
Name: 3	Blinky1		
Location: (4)	E:#Microcontrollers#ATmega328P_XMINI#Wc	orkspace#	Browse
Solution name:	Blinky1	(5)	Create directory for solution
			6 OK Cancel

### Create an AVR Application in Assembly Language (4)

					Device Jafe:
Name	App./Boot Mem	ory (Kbytes)Data Memory	(bytes)EEPROM (b	ytes)	Device Into:
AT90CAN128	128	4096	4096	*	
AT90CAN32	32	2048	1024	=	
AT90CAN64	64	4096	2048		
AT90PWM1	8	512	512		
AT90PWM161	16	1024	512		
AT90PWM216	16	1024	512		
AT90PWM2B	8	512	512		
AT90PWM316	16	1024	512		
AT90PWM3B	8	512	512		No device colocted
AT90PWM81	8	256	512		No device selected
AT90USB1286	128	8192	4096		
AT90USB1287	128	8192	4096		
AT90USB162	16	512	512		
AT90USB646	64	4096	2048		
AT90USB647	64	4096	2048		
AT90USB82	8	512	512		
ATA5272	8	512	512		
ATA5505	16	512	512		
ATA5702M322	64	1024	2304	Ψ.	

	Device Selection						×
2	Device Family Name <u>ATmega328P</u> ATmega328PB Typ ma	y: All App./Boot Memory (Kbytes) 32 32 32 pe the target ake the devic	Data Memory (bytes 2048 2048 chip nar e list sho	DEEPROM (bytes) 1024 1024	ſ	1       328P         Device Info:	
						medbeg	-
						Зок	Cancel

### Create an AVR Application in Assembly Language (5)

Auto-generated source program.



### Create an AVR Application in Assembly Language (6)

Diala di Atasal Chudia (Adasia)

			File Edit View VAssistY ASE Project Build Debug Tools Window Help	
Edit	source file: m	ain.asm	$\begin{array}{c c c c c c c c c c c c c c c c c c c $	<ul> <li>Debug Browser ▼</li> <li>↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓</li></ul>
			Blinky1 ATmega328PB Xplained Mini - 2072 main.asm* 😐 🗙	▼ Solution Explorer ▼ ₽ ×
START:	LDI R16,(1 << 5) OUT DDRB,R16 SBI PINB,5	; Set PB5 as OUTPUT mode ; Toggle LED @PB5	; ; Blinky1.asm ; ; Created: 2016-08-26 오후 10:58:31 ; Author : admin	C      C     C      C      C      C      C      C      C      C
DLY0: DLY1: DLY2: DLY3:	LDI R18,13 LDI R17,125 LDI R16,0 NOP DEC R16 BRNE DLY3 DEC R17 BRNE DLY2 DEC R18 BRNE DLY1 RJMP LOOP		<pre>START: LDI R16,(1 &lt;&lt; 5) ; Set PB5 as OUTPUT mode OUT DDRB,R16</pre> LOOP: SBI PINB,5 ; Toggle LED @PB5DLY0: LDI R18,13 DLY1: LDI R17,125 DLY2: LDI R16,0 DLY3: NOP DEC R16 BRNE DLY3 DEC R17 BRNE DLY2 DEC R18 BRNE DLY1 RJMP LOOP	<ul> <li>▶ Isolatels</li> <li< td=""></li<></ul>
			100 % - 4	*

Biomedical Engineering, Inje University

*ρ* \_ **π** ×

V Quick Launch (Ctrl+O)

### Create an AVR Application in Assembly Language (7)



### Build (Assemble and Link)

or



### Create an AVR Application in Assembly Language (8)

### Build results (succeeded case)

Show output from:       Build       Image: Second S	Output	<b>→</b> ↓ ×				
D:#Atmel#Studio#7.0#toolchain#avr8#avrassembler#avrasm2.exe -fl -o "Blinky1.hex" -m "Blinky1.map" -l "Blinky1.lss' AVRASM: AVR macro assembler 2.1.57 (build 18 Jan 23 2015 13:50:29) Copyright (C) 1995-2015 ATMEL Corporation [builtin](2): Including file 'D:/Atmel#Studio#7.0#Packs#atmel#ATmega_DFP#1.0.90#avrasm#inc#m328PBdef.inc' "ATmega328PB" memory use summary [bytes]: Segment Begin End Code Data Used Size Use% 	Show output from: Build -   🖕   🖕 🛬 🛓 🚈					
Segment         Begin         End         Code         Data         Used         Size         UseX	D: A\ Co [t	::#Atmel#Studio#7.0#toolchain#avr8#avrassembler#avrasm2.exe -fl -o "Blinky1.hex" -m "Blinky1.map" -l "Blinky1.lss' VRASM: AVR macro assembler 2.1.57 (build 18 Jan 23 2015 13:50:29) opyright (C) 1995-2015 ATMEL Corporation builtin](2): Including file 'D:/Atmel#Studio#7.0#Packs#atmel#ATmega_DFP#1.0.90#avrasm#inc#m328PBdef.inc' ATmega328PB" memory use summary [bytes]:				
[.cseg] 0x000000 0x00001c 28 0 28 32768 0.1% [.dseg] 0x000100 0x000100 0 0 0 2048 0.0% [.eseg] 0x000000 0x000000 0 0 0 1024 0.0% Assembly complete, 0 errors. 0 warnings	Se 	egment Begin End Code Data Used Size UseX				
Done executing task "RunAssemblerTask". Done building target "CoreBuild" in project "Blinky1.asmproj". Target "PostBuildEvent" skipped, due to false condition: ('\$(PostBuildEvent)' != '') was evaluated as ('' != ''). Target "Build" in file "D:#Atmel#Studio#7.0#Vs#Avr.common.targets" from project "E:#Microcontrollers#ATmega328P_XMINI#Worksp Done building target "Build" in project "Blinky1.asmproj". Done building project "Blinky1.asmproj".	[ [ As Done to Done build Target "Po Target "Bu Done build Done build	.cseg] 0x000000 0x00001c 28 0 28 32768 0.1% .dseg] 0x000100 0x000100 0 0 0 0 2048 0.0% .eseg] 0x000000 0x000000 0 0 0 0 1024 0.0% .ssembly complete, 0 errors. 0 warnings executing task "RunAssemblerTask". ding target "CoreBuild" in project "Blinky1.asmproj". tostBuildEvent" skipped, due to false condition; ('\$(PostBuildEvent)' != '') was evaluated as ('' != ''). suild" in file "D:#Atmel#Studio#7.0#Vs#Avr.common.targets" from project "E:#Microcontrollers#ATmega328P_XMINI#Worksp ding target "Build" in project "Blinky1.asmproj".				
Build succeeded. Build: 1 succeeded or up-to-date, 0 failed, 0 skipped	Build such	ceeded. = Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ===================================				
	Build succeed					

### Debugging the Application (1)



or



# Debugging the Application (2)

Blinky - AtmelStud       File     Edit       View     View       ○ - ○     10 - 10       ▶ ○     10 - 10       ▶ ○     0 - 10       ▶ ○     0 - 10	lio (Administrator) AssistX ASF Project Build Debug Tools Window Help Tools Window Help Tools Window Help Tools Window Help Tools C Tools Window Help Tools C Tools Window Help Tools C Tools Window Help Tools C Tools C Too
Build Build Events Toolchain Device Tool Components Advanced	Configuration: N/A   Platform: N/A  Selected debugger/programmer  External programming tool  Program using external tool Command:
	Atmel Studio

You will get this error message if you start execution of the application without selecting a debugger/programmer.

Atmel Stud	io 🛛 🚬
1	Please select a connected tool and interface and try again.
	Continue

# Debugging the Application (3)

Build Build Events Toolchain Device 1 Components Advanced 3 Configuration: N/A Selected debugger/progr. Simulator MEDBG ATML2323040200010528				
Selecting a debugger/programmer				

Blinky* 👳 🗙 ATmega32	8P Xplained Mini - 0528 main.c
Build Build Events	Configuration: N/A    Platform: N/A
Toolchain Device Tool Components Advanced	Selected debugger/programmer MEDBG • ATML2323040200010528   Interface: ISP debugWIRE 4
	Programming settings Erase entire chip Preserve EEPROM
	External programming tool  Program using external tool Command:
	Debug settings           Image: Cache all flash memory except

### Debugging the Application (4)



Biomedical Engineering, Inje University

### Debugging the Application (5)



### Open I/O window for debugging (Watching registers associated to PORTB)

I/O	<b>-</b> □ ×
Filter:	-   🚄
Name	Value
AD_CONVERTER ANALOG_COMPARA CPU CPU EPROM CPURENAL_INTERRU	NTOR PT
PORTC	-
Name Address Va	ue Bits
Image         PINB         0x23         0x           Image         DDRB         0x24         0x           Image         PORTB         0x25         0x	98 • • • • • • • • • • • • • • • • • • •

### Debugging the Application (6)

	⊪ ∳ ∎   →		60 🐮 🔽 🏌 🚡   Hex 🕴	%   🔤 - 🚚 💷 📟 🗰 🛎 🛎	₩.	Step Over		
Disa	issembly	main	asm 🕘 🗙 Step Over (F10)			•		
•>	START:	LDI	R16,(1 << 5) ; Set	PB5 as OUTPUT mode				
		OUT	DDRB,R16					
					1/0	<u></u>		- = ×
						Filter:	- I <u>2</u>	
			68 : ? : N I Hex 🕅	8   🔤 - 🚚 🚅 📾 🖨 🔤 🚛 🚢 🕍   (		Name	Value	
Disa	ssembly	main.	asm 🕂 🗙			ANALOG_COMPARATO	R	
	START:	LDI	R16,(1 << 5) ; Set (	PB5 as OUTPUT mode				
		OUT	DDRB,R16			VO PORTB		
	LOOP	CPT				VO PORTC		-
	LUUP:	2D1	FIND, 5 ; logg.	TE LED @PBS		Name Address Value	Bits	
						DDRB 0x24 0x00		

10 PORTB 0x25 0x00

# Debugging the Application (7)



### Debugging the Application (8)



# Debugging the Application (9)



### Debugging the Application (10)

			⇔:?:*	🛨   Hex 🄏   🖼 🗸 💭 🞼 🎟 🛱 👹 🕻
Dis	assembly	mair	Continue (F5)	
	START:	LDI OUT	R16,(1 << 5) DDRB,R16	; Set PB5 as OUTPUT mode
	LOOP:	SBI	PINB,5	; Toggle LED @PB5
	DLY0:	LDI	R18,13	
	DLY1:	LDI	R17,125	
	DLY2:	LDI	R16,0	



LED blinking continues...



Stop debugging

### Debugging the Application (11)



### Start the Application Without Debugging



or



### Congratulations!

You have successfully developed an AVR application in assembly language.



# Create an AVR Application in C Language (1)

- 1. Connect ATmega328PB Xplained Mini to PC via USB Micro-B cable.
- 2. Three drivers will be installed when the board is connected for the first time.

进 드라이버 소프트웨어 설치	×
장치 드라이버 소프트웨어 설치	
USB Composite Device USB 입력 장치 mEDBG CDC	✔사용 준비 완료 ✔사용 준비 완료 ✔Windows Update 검색 중
Windows Update에서 장치 드라이버 소프트워 Windows Update의 드라이버 소프트웨어 다운	ll어를 다운로드하려면 시간이 걸릴 수 있습니다. 2 <u>로드 건너뛰기</u>
	탈기(C)

# Create an AVR Application in C Language (2)

- 1. Run AVR Studio 7.
- Image of the detected ATmega328PB
   Xplained Mini is displayed with related information.



# Create an AVR Application in C Language (3)

1. Select

File – New – Project...



### Create an AVR Application in C Language (4)



### Create an AVR Application in C Language (5)

	Anna (Danak Manana (Mahada	Data Managari da dar			Device Info:	
lame	App./Boot Memory (Kbyte	s)Data Memory (bytes	EEPROM (bytes)		Device Inio.	
4T90CAN128	128	4096	4096	<b>*</b>		
AT90CAN32	32	2048	1024	=		
AT90CAN64	64	4096	2048			
T90PWM1	8	512	512			
AT90PWM161	16	1024	512			
T90PWM216	16	1024	512			
AT90PWM2B	8	512	512			
T90PWM316	16	1024	512			
T90PWM3B	8	512	512		No device selected	
T90PWM81	8	256	512		no device selected	
T90USB1286	128	8192	4096			
T90USB1287	128	8192	4096			
T90USB162	16	512	512			
T90USB646	64	4096	2048			
AT90USB647	64	4096	2048			
T90USB82	8	512	512			
TA5272	8	512	512			
TA5505	16	512	512			
TA5702M322	64	1024	2304	*		

	Device Selection Device Family:	All			<ol> <li>328P</li> </ol>		×
2	Name ATmega328P ATmega328PB Typ mak	App./Boot Memory (Kbytes 32 32 e the target ke the devic	Data Memory (byte 2048 2048 t chip nar ce list shc	s) EEPROM (bytes) 1024 1024 me to ort.	evice Info: Device Name: ATmega Speed: N/A Vcc: N/A Family: ATmega Datasheets Supported Tools Atmel-ICE AVR Dragon AVR Dragon AVR ONEI EDBG JTAGICE3 JTAGICE mkll mEDBG	3 <u>28PB</u>	* U
					3	ок	Cancel

### Create an AVR Application in C Language (6)

Auto-generated source program.

파일은 편집은 보기() VASistX ASF 프로젝트() 빌드() 디버그() 도구() 정() 도울일() 이 · 이 전· 렌 · · · · · · · · · · · · · · · · ·	👸 Blinky - AtmelStudio (관리자) Advanced Mode 🔻 빠른 실행(Ctrl+Q) 🔎 🗕 🗖						
・・・・・・・・・・・・・・・・・・・・・・・・・	파일(F) 편집(E) 보기(V) VAssistX ASF 프로젝트(P) 빌드(B) 디버그(D) 도구(T) 창(W) 도움말(H)						
Na 高 → I → A : ? * I 16전수 名 岡 - 美信 回時 岡 国 : 函 函 函 梁 ● ATTRega322893 * No Tool :          mainc       - ○ → D #atmel_workspace#Blinky#Blinky#mainc       ● ○ △ ○ → ○ ◎ ○ → ○ ◎ ○ → ○         1       E/*       Definition       ●         1       E/*       Definition       ●         2       * Blinky.c       ●       ● ○ △ ○ → ○ ◎ ○ → ○         3       *       ●       ●       ●         4       * Created: 2017-05-30 오전 10:47:52       ●       ● ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○         5       * Author : Administrator       ●       ●       ●         6       */       */       ●       ●       ●       ●       ●       ●         10       ● </td <td>📗 🖸 - 🎯 📩 🔁 🖆 - 😩 🔛 📽 👗 🗗 🏦 💙 - 연 - 🔡 🕄 🕨 Debug 🔹 브라우저 디버그 -</td> <td>- 🗸 🔎 🔎 👘</td>	📗 🖸 - 🎯 📩 🔁 🖆 - 😩 🔛 📽 👗 🗗 🏦 💙 - 연 - 🔡 🕄 🕨 Debug 🔹 브라우저 디버그 -	- 🗸 🔎 🔎 👘					
mainc       ★ Atmega22898 Xplained Mini - 1585       ● #4 월 471       ● #4 월 471	🔡 💹 🖞 🔳 🖹 → 🗉 🕨 📩 🚓 🍸 🕆 🔭 🍸 16진수 🧏 📓 - 😳 頃 💷 🗒 💭 💭 🏭 🚵 🖄 💭 🛫 🗰 A1	Tmega328PB   No Tool 🖕					
mainc       ・ ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・	main.c 😐 🗙 ATmega328PB Xplained Mini - 1585						
1 C/* [2] (* Blinky.c * Blinky.c * Created: 2017-05-30 오전 10:47:52 * Author : Administrator */ * #include <avr io.h=""> 9 10 11 Dint main(void) 12 { 13 /* Replace with your application code */ 14 while (1) 15 { 16 } 19</avr>	→ main.c   → D:#atmel_workspace#Blinky#Blinky#main.c	▼ ኛ Go 🔹 🗠 🖆 👘 🐨 👘 🕨 💻					
2 * Blinky.c * * Created: 2017-05-30 오전 10:47:52 * Author : Administrator */ * #include <avr io.h=""> 9 10 11 Bint main(void) 12 { /* Replace with your application code */ while (1) 15 { 16 } 19</avr>		★ 솔루션 탐색기 검색(Ctrl+;) 🔎 ◄					
3 * 4 * Created: 2017-05-30 오전 10:47:52 5 * Author : Administrator 6 */ 7 8 #include <avr io.h=""> 9 10 11 ₽int main(void) 12 { 13 /* Replace with your application code */ 14 while (1) 15 { 16 } 19</avr>	2 * Blinky.c						
4 * Created: 2017-05-30 오전 10:47:52 * Author : Administrator 6 */ 7 8 #include <avr io.h=""> 9 10 11 Pint main(void) 12 { 13 /* Replace with your application code */ 14 while (1) 15 { 16 } 19</avr>	3 *	a Dependencies					
<pre>5 * Author : Administrator 6 */ 7 8 #include <avr io.h=""> 9 10 11 Dint main(void) 12 { 13 /* Replace with your application code */ 14 while (1) 15 { 16 } 17 } 18 19</avr></pre>	4 * Created: 2017-05-30 오전 10:47:52	🖾 Output Files					
6 [ */ 7 8 #include <avr io.h=""> 9 10 11 Dint main(void) 12 { 13 /* Replace with your application code */ 14 while (1) 15 { 16 } 17 } 18 19</avr>	5 * Author : Administrator	District Sector Sect					
7 8 #include <avr io.h=""> 9 10 11 ■int main(void) 12 { 13 /* Replace with your application code */ 14 while (1) 15 { 16 } 17 } 18 19</avr>	6 */						
8 #include <avr io.h=""> 9 10 11 Dint main(void) 12 { 13 /* Replace with your application code */ 14 while (1) 15 { 16 } 17 } 18 19</avr>	7						
9 10 11 Dint main(void) 12 { 13 /* Replace with your application code */ 14 while (1) 15 { 16 } 17 } 18 19	8 #include <avr io.h=""></avr>						
10 11 ⊕int main(void) 12 { 13 /* Replace with your application code */ 14 while (1) 15 { 16 } 17 } 18 19	9						
11       int main(void)         12       {         13       /* Replace with your application code */         14       while (1)         15       {         16       }         17       }         18	10						
12       {         13       /* Replace with your application code */         14       while (1)         15       {         16       }         17       }         18	11 pint main(void)	VA View VA Outline 술루션 탐색기					
<pre>13 /* Replace with your application code */ 14 while (1) 15 { 16 } 17 } 18 19</pre>	12 {						
14       while (1)         15       {         16       }         17       }         18	13 /* Replace with your application code */	· · · · · · · · · · · · · · · · · · ·					
15     {       16     }       17     }       18	14 while (1)						
16 } 17 _} 18	15 {						
17 _} 18 19	16 }						
18 <b></b> 19	17 }						
19	18						
	19						
131 % -	131.94 - 4	• •					

### Create an AVR Application in C Language (7)



Advanced Mode **▼** 빠른 실행(Ctrl+Q)

o o 🏠 To - 🗗 🖻 🗡 🗕

鸁 솔루션 'Blinky' (1개 프로젝트)

VA View VA Outline 솔루션 탐색기

2 ZJ &

솔루션 탐색기 검색(Ctrl+;)

Dependencies

Output Files

Libraries

c) main.c

Blinky

- 1 🚚

- CGC

₽ \_ □ ×

Q

• 🗊 🖋 🕲 🖬 🗀 • 🗉 🚈 🚈

Biomedical Engineering, Inje University

### Create an AVR Application in C Language (8)



Build (Compile and Link)

or



### Create an AVR Application in C Language (9)

### Build results (succeeded case)

출력	
출력 보기 선	!택(S): 빌드 -   월 ▲   월 ▲   폴   禪
<avrgcc.  "Blinky.cp 조건이 잘듯 "D:♥atmel_ "Blinky.cp "Blinky.cp</avrgcc. 	assembler.general.includePaths> assembler.debugging.DebugLevel>Default (-Wa,-g) Program Memory Usage : 254 bytes 0.8 % Full Data Memory Usage : 0 bytes 0.0 % Full tputFileVerifyTask" 작업을 실행했습니다. rroj" 프로젝트에 "CoreBuild" 대상을 빌드했습니다. 문되어 "PostBuildEvent" 대상을 건너뜁니다. (`\$(PostBuildEvent)' != ``)이(가) (`` != ``)(으)로 확인되었습니다. workspace#Blinky#Blinky#Blinky.cproj" 프로젝트의 "D:#Program Files#Atmel#Studio#7.0#Vs#Avr.common.targets" 파일에 있는 "Build" 대상(진입점): rroj" 프로젝트에 "Build" 대상을 빌드했습니다.
빌드했습니 ======= ↓	다. :빌드: 성공 또는 최신 상태 1, 실패 0, 생략 0 =========
출력	
빌드했습니다.	
### Download and Run the Application

💑 Blinky - AtmelStudio (관리자)			
파일(F) 편집(E) 보기(V) VAssistX ASF 프로젝트(P) 빌드(B)	디바	버그(D) 도구(T) 창(W) 도움말(	(H)
🖸 - ७ - ९   📅 - 🖽 🖆 - 🖆 💾 🚜 🗗 🗇   ७ - ९ -   🔚		창(W)	•
🕅 🛅 🔳   🤿 🗉 🕨   🚓 🏌 🖓 🏌 🛣 🚺 16진수 🄏 📓 🕶	<b>Þ</b> 00	Start Debugging and Break	Alt+F5
main c -ti X ATmega328PB Xolained Mini - 1585		Attach to Target	
→ main c		디버깅 중지(E)	Ctrl+Shift+F5
		디버그하지 않고 시작(H)	Ctrl+Alt+F5
1 =/*		Disable debugWIRE and Close	
2 * Blinky.c	•	Continue	F5
3 *	÷	Execute Stimulifile	
4 * Created: 2017-05-30 오전	€	Set Stimulifile	
5 * Author : Administrator	ΰ	다시 시작(R)	
	63	간략한 조사식(Q)	Shift+F9



or

## Debugging the Application (1)



or



# Debugging the Application (2)

Blinky - AtmelStud       File     Edit       ○ - ○       २ - ●       ▶ ● ●     ● = ●	lio (Administrator) AssistX ASF Project Build Debug Tools Window Help 1 - 2 ■ ■ ■ 从 □ □ □ - 7 - 7 - 1 ■ Q   ▶ ₩ Debug - Debug Browser - ▶   ↔ : 7 : ★ E   Hex ※   ■ - 2 :
Build Build Events Toolchain Device Tool Components Advanced	ZoP Aplained Will - 0526       Haint         Configuration:       N/A         Selected debugger/programmer         Selected debugger/programmer         External programming tool         Program using external tool         Command:
	Atmel Studio

You will get this error message if you start execution of the application without selecting a debugger/programmer.

Atmel Stud	io 🛛 🔍
1	Please select a connected tool and interface and try again.
	Continue

# Debugging the Application (3)

Blinky* 👳 🗙 ATmega32	28P Xplained Mini - 0528			
Build Build Events Toolchain Device 1 Tool Components Advanced	Configuration: N/A Selected debugger/progr 2 Simulator MEDBG ATML2323040200010528			
Selecting a debugger/programmer				

Blinky* → × ATmega328	P Xplained Mini - 0528 main.c
Build Build Events	Configuration: N/A    Platform: N/A
Toolchain Device Tool Components Advanced	Selected debugger/programmer MEDBG • ATML2323040200010528  Interface: ISP debugWIRE (4)
	Programming settings Erase entire chip Preserve EEPROM
	External programming tool  Program using external tool Command:
	Debug settings Keep timers running in stop mode Cache all flash memory except

## Debugging the Application (4)



## Debugging the Application (5)

Blinky1 (Debugging) - AtmelStudio (Administrator)	(1)		Quick Laund	h (Ct	rl+Q)		×
File Edit View VAssistX ASF Project Build	Deb	oug Tools Window Help					
🕴 🖸 🗸 🗿 👘 📲 🎽 🖉 🛱 🖒 👘		Windows 2	•	•	Breakpoints		Ctrl+Alt+B
🛛 🖄 ■ 🕒 → II 🕨 🔂 🛊 🖓 🏌 🔭 Hex	ÞII	Start Debugging and Break			Data Breakpoints		
	Ř	Attach to Target		<b>W</b>	Processor Status		
		Stop Debugging	Shift+F5	1	I/O	3	
	⊳	Start Without Debugging	Ctrl+F5		Live Watch		

#### Open I/O window for debugging (Watching registers associated to PORTB)



## Debugging the Application (6)



## Debugging the Application (10)



#### Congratulations!

You have successfully developed an AVR application in C language.



### What's next?



# Summary

- ATmega328PB instruction set
- Assembly language directives
- Assembly language expressions
- Mixed Language Programming







-