ภาษาแอสเซมบลี้ของ AVR

(AVR Assembly Language)

โดย

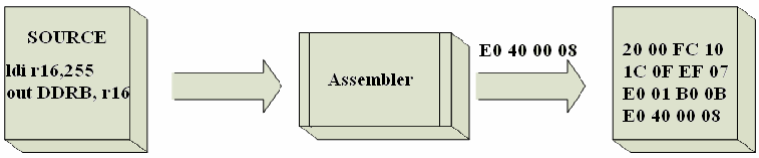
รศ.ฌรงค์ บวบทอง

ภาควิชาวิศวกรรมไฟฟ้าและคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

มหาวิทยาลัยธรรมศาสตร์

Assembler



ลักษณะภาษาแอสเซมบลี้

```
/*
 * AVRAssembler1.asm
 *
 *  Created: 9/7/2554 9:40:18
 *   Author: xp
 */
        .org 0
        rjmp    RESET           ;Reset Handle
        rjmp    RESET
        rjmp    RESET
RESET:  ldi     r16, 0b11111111 ;load register r16 with all 1's
        out     DDRB, r16       ;configure PORT B for all outputs
        ldi     r16, 0b00000000 ;load register r16 with all 0's
        out     DDRD, r16       ;configure PORTD for all inputs
loopit: in      r16, PIND       ;read the state of the pin on PORTD
        out     PORTB, r16      ;and copy it to PORTB
        rjmp loopit
```

3

ลักษณะภาษาแอสเซมบลี้

1. ฟิลด์เลเบล (Label field)
2. ฟิลด์คำสั่งตัวแปลภาษา (Assembler directives)
3. ฟิลด์คำสั่ง (Instruction field)
4. ฟิลด์ตัวกระทำ (Operands field)
5. ฟิลด์หมายเหตุ (Comment field)

4

ภาษาแอสเซมบลี้ (Assembly Language)

ในแต่ละบรรทัดอาจประกอบด้วย

[label:]	directive	[operands]	[Comment]
[label:]	instruction	[operands]	[Comment]
Comment			
Empty line			

โดยมีความยาวไม่เกิน 120 ตัวอักษร (ขึ้นอยู่กับ Assembler)

เลเบลฟิลด์ (label field)

ก) ชื่อเลเบลควรมีความหมายตรงกับงานที่ทำ
ข) เป็นตัวอักษร A ถึง Z (เป็นตัวพิมพ์ใหญ่หรือตัวพิมพ์เล็กก็ได้)
ค) เป็นตัวเลข 0 ถึง 9
ง) เป็นสัญญลักษณ์พิเศษบางตัวเช่น @ _ เป็นต้น (แต่ควรพยายามหลีกเลี่ยง)
จ) จะประกอบด้วยตัวอักษรจากข้อ 1 ถึง 3 อย่างไรก็ได้แต่ควรขึ้นต้นด้วยตัวอักษรในข้อ ข) เสมอ
ฉ) เลเบลต้องตามหลังด้วยโคลอน (:) เสมอ (ตัวแปลบางตัว ยกเว้นที่ใช้คู่กับคำสั่ง EQU)
ช) ห้ามตั้งชื่อซ้ำกับรหัสนีโมนิคของซีพียู หรือ Directive

ฟิลด์คำสั่งของตัวแปลภาษา (assembler directives field)

ตัวแปลภาษาแอสเซมบลี้ มีคำสั่งอยู่ 2 แบบ

- คำสั่งของไมโครโปรเซสเซอร์
- คำสั่งของตัวแปลภาษา

คำสั่งแบบหลังนี้ จะไม่ถูกนำไปใช้ กับไมโครโปรเซสเซอร์ เพราะมันไม่ได้ถูกแปลเป็นภาษาเครื่อง แต่มีใช้เพื่ออำนวยความสะดวกให้ผู้ใช้สำหรับการ เขียนโปรแกรมเช่นการกำหนดค่าให้กับตัวแปร จองที่อยู่และกำหนดค่าในหน่วยความจำ รวมถึงการกำหนดตำแหน่งที่อยู่ของโปรแกรม เช่น DB, DW และ DD แต่คำสั่งเหล่านี้มิได้มีผลโดยตรงต่อค่าหน่วยความจำในขณะไมโครโปรเซสเซอร์ทำงาน คำสั่งเหล่านี้เพียงแต่ กำหนดค่าต่างๆ และเพิ่มเติมข่าวสารบางอย่างลงในออปเจ็กต์ไฟล์เพื่อประโยชน์ในการดีบักเท่านั้น

ตัวอย่างคำสั่งของตัวแปลภาษา

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

คำสั่งเหล่านี้เวลาใช้จ้องมี . (จุดนำหน้า) เช่น

.ORG 0

คำสั่งของตัวแปลภาษา INCLUDE

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 EXIT directive is encountered. An included file may itself contain INCLUDE directives.

Syntax:

.INCLUDE “filename”

Example:

```
                                ; iodefs.asm:
.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
```


คำสั่งของตัวแปลภาษา BYTE

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.

Syntax:

LABEL: .BYTE expression

Example:

```
.DSEG
        var1:   .BYTE 1             ; reserve 1 byte to var1
        table:  .BYTE tab_size      ; reserve tab_size bytes
.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
```

คำสั่งของตัวแปลภาษา DB

DB-Define constant byte(s) in program memory or E2PROM 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.

Syntax:

```
LABEL: .DB expressionlist
```

Example:

```
.CSEG
        consts: .DB 0, 255, 0b01010101, -128, 0xaa
.ESEG
        eeconst:.DB 0xff
```


คำสั่งของตัวแปลภาษา DW

DW-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 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 -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.

Syntax:

```
LABEL: .DW expressionlist
```

Example:

```
.CSEG
        varlist:    .DW 0,0xffff,0b1001110001010101,-32768,65535
.ESEG
        eevar:      .DW 0xffff
```

คำสั่งของตัวแปลภาษา DEF

DEF - Set a symbolic name on a register

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

Syntax:

```
.DEF    Symbol=Register
```

Example:

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


คำสั่งของตัวแปลภาษา CSEG

CSEG - Code Segment

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

Syntax:

```
.CSEG
```

Example:

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

คำสั่งของตัวแปลภาษา DSEG

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.

Syntax:

```
.DSEG
```

Example:

```
.DSEG                                   ; Start data segment
var1:   .BYTE   1                       ; reserve 1 byte to var1
table:  .BYTE   tab_size                ; reserve tab_size bytes.
.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
```

15

คำสั่งของตัวแปลภาษา ESEG

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.

Syntax:

```
.ESEG
```

Example:

```
.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
```

16

คำสั่งของตัวแปลภาษา EXIT

EXIT - Exit this file

The EXIT directive tells the Assembler to stop assembling the file. Normally, the Assembler runs until end of file (EOF). If an EXIT directive appears in an included file, the Assembler continues from the line following the INCLUDE directive in the file containing the INCLUDE directive.

Syntax:

```
.EXIT
```

Example:

```
.EXIT                    ; Exit this file
```


คำสั่งของตัวแปลภาษา LIST และ NOLIST

LIST - Turn the listfile generation on
NOLIST - Turn listfile generation off

The LIST directive tells the Assembler to turn listfile generation on. The Assembler generates a listfile which is a combination of assembly source code, addresses and opcodes. Listfile generation is turned on by default. The directive can also be used together with the NOLIST directive in order to only generate listfile of selected parts of an assembly source file.

Syntax:

```
.LIST
```

Example:

```
.NOLIST                          ; Disable listfile generation
.INCLUDE     “macro.inc”         ; The included files will not
.INCLUDE     “const.def”         ; be shown in the listfile
.LIST                            ; Reenable listfile generation
```

คำสั่งของตัวแปลภาษา MACRO และ ENDMACRO

MACRO - Begin macro
The MACRO directive tells the Assembler that this is the start of a Macro. The MACRO directive takes the Macro name as parameter. When the name of the Macro is written later in the program, the Macro definition is expanded at the place it was used. A Macro can take up to 10 parameters. These parameters are referred to as @0-@9 within the Macro definition. When issuing a Macro call, the parameters are given as a comma separated list. The Macro definition is terminated by an ENDMACRO directive. By default, only the call to the Macro is shown on the listfile generated by the Assembler. In order to include the macro expansion in the listfile, a LISTMAC directive must be used. A macro is marked with a + in the opcode field of the listfile.

Syntax:

```
.MACRO macroname
```

ENDMACRO - End macro
The ENDMACRO directive defines the end of a Macro definition. The directive does not take any parameters. See the MACRO directive for more information on defining Macros.

Syntax:

```
.ENDMACRO
```


ตัวอย่าง MACRO และ ENDMACRO

Example:

```
.MACRO SUBI16                   ; Start macro definition
        subi    @1, low(@0)     ; Subtract low byte
        sbci    @2,high(@0)     ; Subtract high byte
.ENDMACRO                       ; End macro definition

.CSEG ; Start code segment
        SUBI16 0x1234,r16,r17   ; Sub.0x1234 from r17:r16
```

เมื่อแทนเป็นโปรแกรมมีความหมายเป็น

```
        Subi    r16,0x34
        sbci    r17,0x12
```

คำสั่งของตัวแปลภาษา LISTMAC

LISTMAC - Turn macro expansion on

The LISTMAC directive tells the Assembler that when a macro is called, the expansion of the macro is to be shown on the listfile generated by the Assembler. The default is that only the macro-call with parameters is shown in the listfile.

Syntax:

```
.LISTMAC
```

Example:

```
.MACRO MACX    ; Define an example macro
       add r0,@0 ; Do something
eor    r1,@1    ; Do something
.ENDMACRO ; End macro definition

.LISTMAC       ; Enable macro expansion
MACX   r2,r1   ; Call macro, show expansion
```


คำสั่งของตัวแปลภาษา DEVICE

DEVICE - Define which device to assemble for

The DEVICE directive allows the user to tell the Assembler which device the code is to be executed on. If this directive is used, a warning is issued if an instruction not supported by the specified device occurs in the code. If the size of the Code Segment or EEPROM Segment is larger than supported by the specified device, a warning is issued. If the DEVICE directive is not used, it is assumed that all instructions are supported and that there are no restrictions on memory sizes.

Syntax:

```
.DEVICE AT90S1200 | AT90S2313 | AT90S4414 | AT90S8515
```

Example:

```
.DEVICE AT90S1200       ; Use the AT90S1200
.CSEG
        push    r30     ; This statement will generate
                        ; a warning since the
                        ; specified device does not
                        ; have this instruction
```

คำสั่งของตัวแปลภาษา EQU

EQU - Set a symbol equal to an expression

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

Syntax:

```
.EQU label = expression
```

Example:

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


คำสั่งของตัวแปลภาษา ORG

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 and
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.

Syntax:

```
.ORG expression
```

ตัวอย่างคำสั่งของตัวแปลภาษา ORG

```
Example:
        .DSEG                       ; Start data segment
        .ORG 0x67                   ; Set SRAM address to hex 67
        variable:.BYTE 1            ; Reserve a byte at SRAM addr. 67H

        .ESEG                       ; Start EEPROM Segment
        .ORG    0x20                ; Set EEPROM location counter
        eevar: .  DW    0xfeff      ; Initialize one word

        .CSEG
        .ORG    0x10                ; Set Program Counter to hex 10
                mov     r0,r1       ; Do something
```


คำสั่งของตัวแปลภาษา SET

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.

Syntax:

```
        .SET label = expression
```

Example:

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

ฟิลด์คำสั่ง (Instruction field)

เป็นคำสั่งที่ผู้ออกแบบซีพียูกำหนด เช่น

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add without Carry	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rd, K	Add Immediate to Word	Rd+1:Rd ← Rd+1:Rd + K	Z,C,N,V	2
SUB	Rd, Rr	Subtract without Carry	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Immediate	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract Immediate with Carry	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rd, K	Subtract Immediate from Word	Rd+1:Rd ← Rd+1:Rd - K	Z,C,N,V	2
AND	Rd, Rr	Logical AND	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND with Immediate	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR with Immediate	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← $FF - Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← $00 - Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • ($FFh - K)	Z,N,V	1

27

ฟิลด์ตัวกระทำ (Operands field)

- **Registers and Operands**
  - Rd: Destination (and source) register in the Register File
  - Rr: Source register in the Register File
  - R: Result after instruction is executed
  - K: Constant data
  - k: Constant address
  - b: Bit in the Register File or I/O Register (3-bit)
  - s: Bit in the Status Register (3-bit)
  - X,Y,Z: Indirect Address Register
  - (X=R27:R26, Y=R29:R28 and Z=R31:R30)
  - A: I/O location address
  - q: Displacement for direct addressing (6-bit)

สัญลักษณ์อีกแบบที่สามารถพบได้

Category	Abbrev.	Means ...	Value range
Register	r1	Ordinary Source and Target register	R0..R31
	r2	Ordinary Source register	
	rh	Upper page register	R16..R31
	rd	Twin register	R24(R25), R26(R27), R28(R29), R30(R31)
	rp	Pointer register	X=R26(R27), Y=R28(R29), Z=R30(R31)
	ry	Pointer register with displacement	Y=R28(R29), Z=R30(R31)
Constant	k63	Pointer-constant	0..63
	c127	Conditioned jump distance	-64..+63
	c255	8-Bit-Constant	0..255
	c4096	Relative jump distance	-2048..+2047
	c65535	16-Bit-Address	0..65535
Bit	b7	Bit position	0..7
Port	p1	Ordinary Port	0..63
	pl	Lower page port	0..31

เกี่ยวกับ I/O Ports

I/O Port ของ AVR แต่ละพอร์ทมีรีจิสเตอร์ 3 ตัวคือ

- DDRx register
- PORTx register
- PINx register

(x หมายถึง พอร์ท A B C และ D)

รีจิสเตอร์ DDRx

ใช้กำหนดทิศทางของพอร์แต่ละบิต โดยถ้าให้บิตใหนเป็น ‘0’ พอร์ทบิตนั้นจะเป็นพอร์ทอินพุท แต่ถ้าให้เป็น ‘1’ จะเป็นพอร์ทเอาท์พุท เช่น

```
ldi     r16, 0XFF          ; load register r16 with all 1’s
out     DDRB, r16          ; พอร์ท B เป็นพอร์ทเอาท์พุททุกบิต
ldi     r16, 0b00000000    ; load register r16 with all 0’s
out     DDRD, r16          ; พอร์ท D เป็นพอร์ทอินพุททุกบิต
```

ภาษาซี

```
DDRB = 0b11111111;      // พอร์ท B เป็นพอร์ทเอาท์พุททุกบิต
DDRD = 0b00000000;      // พอร์ท D เป็นพอร์ทอินพุททุกบิต
DDRA = 0b00001111;      // พอร์ท A บิต 0 – 3 เป็นเอาท์พุท  บิต 4-7 เป็นอินพุท
```


รีจิสเตอร์ **PINx** (Port IN)

ใช้สำหรับการอ่านข้อมูลจากขาพอร์ท เช่น
ถ้าต้องการอ่านข้อมูลจากพอร์ท A

```
ldi     r16, 0b00000000   ; โหลดรีจิสเตอร์ r16 ด้วย ‘0’ ทั้งหมด
out     DDRD, r16         ; พอร์ท D เป็นพอร์ทอินพุททุกบิต
in      r16,PIND          ; อ่านข้อมูลจากพอร์ท D
```

ภาษาซี

ถ้าต้องการอ่านข้อมูลจากพอร์ท A

```
DDRA = 0x00;      //  Set port a as input
x = PINA;         // Read contents of port a
```

รีจิสเตอร์ **PORTx**

ทำหน้าที่ได้ 2 แบบ

1. ทำหน้าที่เป็นพอร์ทเอาท์พุท โดยกำหนดให้ DDRx เป็น ‘1’ เช่น

```
ldi  r16, 0b11111111   ; โหลดรีจิสเตอร์ r16 ด้วย ‘0’ ทั้งหมด
out DDRD, r16                  ; พอร์ท D เป็นพอร์ทเอาท์พุททุกบิต
out PORTD, r15                 ; ส่งข้อมูลจาก r15 ออกพอร์ท D
```

ภาษาซี

ถ้าต้องการส่งข้อมูลออกทางพอร์ท D

```
DDRD = 0x00;     //  Set port a as output
PORTD = 25;      // ส่ง 25 ออกทางพอร์ท D
DDRC.0 = 1;      // กำหนดให้พอร์ท c บิต 0 เป็นเอาท์พุทเท่านั้น
PORTC.0 = 1;     // ทำให้พอร์ท C บิต 0 เป็น high หรือ ‘1’
```

รีจิสเตอร์ **PORTx**

2. เมื่อกำหนดให้ DDRx เป็น ‘0’ จะทำให้พอร์ทเป็นพอร์ทอินพุท และสามารถกำหนดสถานะของขาของพอร์ทนี้ได้ด้วยการกำหนดค่าให้แก่รีจิสเตอร์ PORTx โดยถ้าให้เป็น ‘1’ จะทำให้ขาพอร์ท มีความต้านทานพูลอับ (Pull up) แต่ถ้าให้เป็น ‘0’ จะทำให้ขาพอร์เป็น Tri state

```
ldi  r16, 0b00000000   ; โหลดรีจิสเตอร์ r16 ด้วย ‘0’ ทั้งหมด
out DDRD, r16          ; พอร์ท D เป็นพอร์ทอินพุททุกบิต
ldi  r16,0b11111111    ; โหลดรีจิสเตอร์ r16 ด้วย ‘0’ ทั้งหมด
out PORTD, r16         ; ทำให้ขาพอร์ท D มีความต้านทานพูลอับ
in   r15,PIND          ; อ่านข้อมูลจากขาพอร์ท D มาที่รีจิสเตอร์ r15
```

รีจิสเตอร์ **PORTx**

ภาษาซี

กำหนดให้พอร์ท A เป็นอินพุทแบบมี pull-ups และอ่านข้อมูลจากพอร์ท A

```
DDRA = 0x00;      // make port a as input
PORTA = 0xFF;     // enable all pull-ups
y = PINA;         // read data from port a pins
```

กำหนดให้พอร์ท B เป็นอินพุทแบบ tri state

```
DDRB  = 0x00;     // make port b as input
PORTB = 0x00;     // disable pull-ups and make it tri state
```

กำหนดให้พอร์ท A 4 บิตล่างเป็นเอาท์พุท แต่ 4 บิตบนเป็นอินพุทแบบมี pull-ups

```
DDRA  = 0x0F;     // lower nib> output, higher nib> input
PORTA = 0xF0;     // lower nib> set output pins to 0,
                  // higher nib> enable pull-ups
```


สรุปการกำหนดค่าให้แก่พอร์ท I/O และผลที่ได้รับ

register bits ⟶ **pin function↓**	**DDRx.n**	**PORTx.n**	**PINx.n**
tri stated input	0	0	read data bit x = PINx.n;
pull-up input	0	1	read data bit x = PINx.n;
output	1	write data bit PORTx.n = x;	n/a

เกี่ยวกับ I/O Ports

Port	Register	Function	Port-Address	RAM-Address
A	PORTA	Data Register	0x1B	0x3B
	DDRA	Data Direction Register	0x1A	0x3A
	PINA	Input Pins Address	0x19	0x39
B	PORTB	Data Register	0x18	0x38
	DDRB	Data Direction Register	0x17	0x37
	PINB	Input Pins Address	0x16	0x36
C	PORTC	Data Register	0x15	0x35
	DDRC	Data Direction Register	0x14	0x34
	PINC	Input Pins Address	0x13	0x33
D	PORTD	Data Register	0x12	0x32
	DDRD	Data Direction Register	0x11	0x31
	PIND	Input Pins Address	0x10	0x30

ฟิลด์หมายเหตุ (Comment field)

เป็นส่วนที่ใช้อธิบายโปรแกรม ฟิลด์หมายเหตุต้องนำหน้าด้วยเครื่องหมายเซมิโคลอน (;) หรืออยู่ระหว่างเครื่องหมาย /* …..*/เสมอ และอาจจะเป็นหมายเหตุทั้งบรรทัด ก็ได้ เช่น

```
/*
 * AVRAssembler1.asm
 *
 *  Created: 9/7/2554 9:40:18
 *   Author: xp
 */
            .org 0
            rjmp    RESET           ;Reset Handle
            rjmp    RESET
            rjmp    RESET
RESET:      ldi     r16, 0b11111111 ;load register r16 with all 1's
            out     DDRB, r16       ;configure PORT B for all outputs
            ldi     r16, 0b00000000 ;load register r16 with all 0's
            out     DDRD, r16       ;configure PORTD for all inputs
loopit:     in      r16, PIND       ;read the state of the pin on PORTD
;into r16 register
            out     PORTB, r16      ;and copy it to PORTB
            rjmp loopit
```

การพัฒนาโปรแกรมภาษาแอสเซมบลี้

39

ภาษาแอสเซมบลี้ (Assembly Language)