



AVR Assembly Fundamentals

ENEE 3582

Microp

Software Levels

High-Level Language	Well structured high-level programming. C++, JAVA, ... etc.
Assembly Language	Low-level programming using English text
Machine Language*	Binary files. Opcode instructions
Instruction Set Architecture	Hardware implementation of opcodes
Microarchitecture	Registers, ALU, ... etc.
Digital Logic	Digital Circuits

Language Translations

English: Display the sum of A times B plus C.

C++: `cout<< (A*B+C) ;`

Assembly Language:

```
ld R1,A
mul B
add R1,C
call WriteInt
```

Machine Language:

```
A1 00000000
F7 25 00000004
03 05 00000008
E8 00500000
```

Assembly vs HLL

- | | |
|--|---|
| <ul style="list-style-type: none"> ❖ Platform (hardware) specific <ul style="list-style-type: none"> ➤ Not portable ❖ Assembled into machine code <ul style="list-style-type: none"> ➤ Translation process ➤ Efficient use of hardware resources ❖ Very little formal structure ❖ Doesn't require OS ❖ Lengthy coding time ❖ Best for hardware control <ul style="list-style-type: none"> ➤ Real-time response ➤ Low-power applications ➤ Low-memory applications | <ul style="list-style-type: none"> ❖ Platform (hardware) independent <ul style="list-style-type: none"> ➤ Very portable ❖ Compiled into machine code <ul style="list-style-type: none"> ➤ Rules and algorithmic process ➤ Inefficient machine code ❖ Formal structure ❖ Requires OS ❖ Faster coding time ❖ Best for app development <ul style="list-style-type: none"> ➤ Fast app development ➤ Powerful apps ➤ Advanced mem use |
|--|---|

Microchip Studio (IDE)

❖ Assembler

- translate the assembly language into a binary file
 - Easier to write code in English
 - Binary files are in machine code
- Supports various Assembly Languages for a many chips (AVR, SAM, U)
- Allow for some complex operations that are not supported in pure assembly
 - If/then/else etc.

❖ Simulator

- Helps test out code

❖ Debugger

❖ Communication

- Download/flash binary to the board

Assembler: Language Elements

1. Constants
2. Reserved words
3. Identifiers
4. Operands
5. Comments

1. Constants: Dec, Bin, Oct, Hex

- ❖ Aka immediates

- ❖ Decimal: Base 10

- no prefix.
- E.g. 10, 123, -567

- ❖ Binary: Base 2

- prefix **0b**.
- E.g. **0b00101110**

- ❖ Hex(adecimal): Base 16

- prefix **0x** or **\$**.
- E.g. **0x1234**, **\$56**

- ❖ Octal: Base 8

- Prefix 0
- E.g. **012**

Constants: ASCII

- ❖ American Standard Code for Information Interchange
- ❖ Each character is a byte value
- ❖ Use quotes for ASCII
- ❖ Table : <https://en.cppreference.com/w/c/language/ascii>
 - '0' - '9' : 0x30 – 0x39
 - 'A' - 'Z' : 0x41 – 0x5A
 - 'a' - 'z' : 0x61 – 0x7A

2. Reserved Words

- a. Operators
- b. Special symbols
- c. AVR Assembly Instructions
- d. Assembler directives
- e. Assembler built-in functions
- f. Preprocessor directives
- g. Preprocessor macros

2a.Reserved Words: Operators

❖ Assembler allows you to write constants as an integer expression

➤ Evaluated by the assembler

❖ Operators:

* multiplication

/ division

% modulus (remainder)

+ addition, positive

- subtraction, negative

>> logical binary shift right

<< logical binary shift left

& bitwise AND

| bitwise OR

^ bitwise XOR

~ bitwise NOT

❖ E.g. $100 \% 5 >> 1 * 3 / 2$

Expressions Order of Precedence

❖ Follows the C rules for operator precedence and type propagation

1. Parenthesis
2. Negative and Positive. Right to Left.
3. Multiplication and Division and Modulus. Left to Right.
4. Addition and Subtraction. Left to Right.
5. Bitwise shift left and shift right. Left to Right.
6. Bitwise AND. Left to Right.
7. Bitwise XOR. Left to Right.
8. Bitwise OR. Left to Right.

https://en.cppreference.com/w/c/language/operator_precedence

2b. Reserved Words: Special Symbols

‘ “ : \ ; /* */ // # .

- ❖ **quotes** used for ASCII strings
- ❖ **Colon** used for labels
- ❖ **Backslash** used for continuation
- ❖ **Semicolon, slash-asterix, double forward slash** used for comments
- ❖ **Pound** for pre-processor directives and functions
- ❖ **Dot** for processor directives

2c. Reserved Words: AVR Instructions

adc	add	adiw	and	andi	asr	bclr	bld	brbc	brbs	brcc
brcs	break	breq	brge	brhc	brhs	brid	brie	brlo	brlt	brmi
brne	brpl	brsh	brtc	brts	brvc	brvs	bset	bst	call	cbi
cbr	clc	clh	cli	cln	clr	cls	clt	clv	clz	com
cp	cpc	cpi	cpse	dec	eicall	eijmp	elpm	eor	fmul	fmuls
fmulsu	icall	ijmp	in	inc	jmp	ld	ldd	ldi	lds	lpm
lsl	lsr	mov	movw	mul	muls	mulsu	neg	nop	or	ori
out	pop	push	rcall	ret	reti	rjmp	rol	ror	sbc	sbc
sbi	sbic	sbis	sbiw	sbr	sbrc	sbrs	sec	seh	sei	sen
ser	ses	set	sev	sez	sleep	spm	st	std	sts	sub
subi	swap	tst	wdr							

2d. Reserved Words: Assembler Directives

❖ 32 Assembler Directives

.byte	.cseg	.csegsize	.db	.dd	.def	.device	.dq
.dseg	.dw	.elif	.else	.endif	.endm	.endmacro	.equ
.error	.eseg	.exit	.if	.ifdef	.ifndef	.include	.list
.listmac	.macro	.message	.nolist	.org	.set	.undef	

2d. Reserved Words: Built-in Functions

abs	byte1	byte2	byte3	byte4
exp2	frac	high	hwrđ	int
log2	low	lwrd	page	pc
q15	q7			

2e. Reserved Words: Preprocessor Directives

#define	#undef	#ifdef	#ifndef
#if	#else	#elif	#endif
#error	#include	#pragma	#
#warning	#message	defined	

2f. Reserved Words: Preprocessor Macros

__AVRASM_VERSION__

__CORE_VERSION__

__DATE__

__FILE__

__LINE__

__PART_NAME__

__TIME__

3. Identifiers

❖ Variables

- Format: variable_name: .type value
- Example: num1: .DB 100

❖ Symbolic constants

- Format: .def constant_name value
- Example .def Kili 1024

❖ Line labels

- Format: line_label:

❖ Function names

- Format: function_name:

❖ Macro names

- Format: .macro macro_name
- .endm

4. Operands

❖ Registers:

- Data register: **R0-R31**
- Index register: **X, Y, Z**
 - Used to access memory
 - Increment: **X+, Y+, Z+**
 - Decrement: **-X, -Y, -Z**

❖ Built-in functions

❖ User-defined constants

❖ Constants

- Values and expressions

5. Comments

- ❖ End of the code line
- ❖ Single line comments: use semicolon and double-forward slash
 - ;
 - //
- ❖ Multiple lines of comments: use asterix-slash
 - /* : marks the beginning of comment section
 - */ : marks the end

Program Template

```
;Assignment number
;Name
;class
;Date
```

```

Variables:      .DSEG          //IRAM
                //indent all code except for identifiers
                //variables
                .CSEG          //flash
                //program      //use CAPS for reserved words
                                //use mixed/lower case for identifiers

loop:           RJMP loop      //infinite loop

program_data:   //initialized values
```