Assembly CodeCount™

Counting Standard

University of Southern California

Center for Systems and Software Engineering

June , 2014

Revision Sheet

Date	Version	Revision Description	Author
6/18/2014	1.0	Original Release	CSSE

Table of Contents

1. Introduction

Unlike high-level languages, assembly languages exist in many variations since they are hardware (architecture) dependent. On top of this, there can be multiple assemblers for a given hardware platform. Unfortunately, these assemblers do not recognize the same assembly-language program on the same platform. In other words, an assembly-language program is architecture dependent and assembler dependent. As a result, there are many variations of assembly languages. It is infeasible to develop an assembly code counter that addresses all existing variations. This assembly code counter aims to address the following assemblers and architectures:

	Assemblers				
	NASM*	**MASM***	**GAS***	**SPIM (MIPS)***	**AIX Assembler (PowerPC)**
Architectures	x86	x86	x86, Motorola, PowerPC	MIPS32	PowerPC, POWER Family (POWER 2, 3, 4, 5, PPC970)

*NASM, MASM, and GAS stand for Netwide Assembler, Microsoft Assembler, and GNU Assembler, respectively. SPIM is a reversal of the letters “MIPS”. SPIM is a simulator designed to run MIPS32 programs.

This code counter supports the following file extensions: *.asm, *.s, and *.asm.pcc.

2. Definitions

2.1. **SLOC –** Source Lines of Code is a unit used to measure the size of software program. SLOC counts the program source code based on a certain set of rules. SLOC is a key input for estimating project effort and is also used to calculate productivity and other measurements.

2.2. **Physical SLOC –** One physical SLOC is corresponding to one line starting with the first character and ending by a carriage return or an end-of-file marker of the same line, and which excludes the blank and comment line.

2.3. **Logical SLOC –** Lines of code intended to measure "statements", which normally terminate by a semicolon (C/C++, Java, C#) or a carriage return (VB, Assembly), etc. Logical SLOC are not sensitive to format and style conventions, but they are language-dependent.

2.4. **Data declaration line or data line –** A line that contains declaration of data and used by an assembler or compiler to interpret other elements of the program. A typical assembly program has two sections or segments, namely the data section and the text (code) section. Lines of codes under the data section are considered data lines. Lines of code under the text section are considered executable lines.

The following table lists the assembly keywords that denote the beginning of a data section for Netwide Assembler (NASM), Microsoft Assembler (MASM), GNU Assembler (GAS), MIPS assembler (SPIM), and PowerPC assembler (AIX):

NASM	MASM	GAS	MIPS	PowerPC
section .data	.data	.data	.data	.data
section .bss	.const	.bss	.rdata	.bss
		.section name, "d"	.sdata	.csect[RW]
		.section name, "b"	.bss	.csect[TC0]
			.sbss	.csect[TC]
			.lit	.csect[TD]
				.csect[UA]
				.csect[DS]
				.csect[BS]
				.csect[UC]
				.csect[ER]
				.csect[SD]
				.csect[LD]
				.csect[CM]

Table 1 Data Declaration Types

2.5. **Compiler Directives –** A statement that tells the compiler how to compile a program, but not what to compile. Assembly program does not contain any compiler directives.

2.6. **Blank Line –** A physical line of code, which contains any number of white space characters (spaces, tabs, form feed, carriage return, line feed, or their derivatives).

2.7. **Comment Line –** A comment is defined as a string of zero or more characters that follow language-specific comment delimiter.

Assembly comment delimiters are "#", ";", "|", and "/*" and are dependent on each assembler. A whole comment line may span one line and does not contain any compliable source code. An embedded comment can co-exist with compliable source code on the same physical line. Banners and empty comments are treated as types of comments.

2.8. **Executable Line of code –** A line that contains software instruction executed during runtime and on which a breakpoint can be set in a debugging tool. An instruction can be stated in a simple or compound form.

- An executable line of code may contain the following program control statements:
  - Compare statements (cmp)
  - Jump statements (jp, exit)
  - Expression statements (assignment statements, operations, system call, etc.)
- An executable line of code may not contain the following statements:
  - Data declaration (data) lines
  - Whole line comments, including empty comments and banners
  - Blank lines

A typical assembly program has two sections or segments, namely the data section and the text section. Lines of codes under the text section are considered executable lines.

The following table lists the assembly keywords that denote the beginning of a text section for Netwide Assembler (NASM), Microsoft Assembler (MASM), GNU Assembler (GAS), MIPS assembler, and PowerPC assembler:

NASM	MASM	GAS	MIPS	PowerPC
section .text	.code	.text	.text	.text
section .txt		.section .text	.init	
			.fini	
			.ktext	

3. Checklist for source statement counts

PHYSICAL SLOC COUNTING RULES			
MEASUREMENT UNIT	ORDER OF PRECEDENCE	PHYSICAL SLOC	COMMENTS
Executable Lines	1	One Per line	Defined in 1.8
Non-executable Lines			
Declaration (Data) lines	2	One per line	Defined in 1.4
Comments			Defined in 1.7
On their own lines	3	Not Included (NI)	
Embedded	4	NI	
Banners	5	NI	
Empty Comments	6	NI	
Blank Lines	7	NI	Defined in 1.6

LOGICAL SLOC COUNTING RULES				
NO.	STRUCTURE	ORDER OF PRECEDENCE	LOGICAL SLOC RULES	COMMENTS
R01	Multiple statements in a single physical line, with each statement delimited by a semicolon	1	Count once per semicolon, excluding empty statement	A physical line (physical SLOC) contains one or more logical statements. Multiple statements are normally delimited by a semicolon. For example, MIPS statements are delimited by a semicolon. In GAS and PowerPC, statements are delimited by a new-line character or a semicolon (unless the semicolon conflicts with the comment character).
R02	A single statement terminated by a new-line character or a semicolon	2	Count once	This is the case where there is only one statement in a physical line.

4. Examples

POWERPC ASSEMBLY

E1 – Example 1: PowerPC Assembly

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT	SLOC TYPE
[label:] mnemonic [operand1[,operand2…]] [# comment]	mylabel: add 6, 4, 5	1	Executable
Multiple statements, each of which is on its own line	add 6, 4, 5 li 0, 1	1 1	Executable Executable
Multiple statements on a single line delimited by ;	add 6, 4, 5; li 0, 1	2	Executable
Label on its own line	start_here: add 6, 4, 5	0 1	 Executable
Label followed by an executable statement	start_here: add 6, 4, 5	1	Executable
Comment lines	#This is a comment line #Another line of comment	0 0	
Statement followed by an inline comment	add 6, 4, 5 #My inline comment	1	Executable
Beginning of a data declaration section	.data #data section begins msg: .string "Hello World"	1 0 1	Data Data
Beginning of a text (code) section	.text # begin code addi 4, 4, msg	1 1	Executable Executable

NETWIDE ASSEMBLY (NASM)

E2 – Example 2: NASM Assembly – A Complete "Hello World" program

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT	SLOC TYPE
Comment lines begin with a semi-colon.	; In NASM, a comment line begins with a semicolon ; hello.asm is a first program for NASM for Linux, Intel, gcc ;	0 0 0	

	; assemble: nasm -f elf -l hello.lst hello.asm	0	
	; link: gcc -o hello hello.o	0	
	; run: hello	0	
	; output is: Hello World	0	
		0	
Start of a data declaration	SECTION .data ; data section	1	Data
section. Lines under this	msg: db "Hello World",10 ; the string to print, 10=cr	1	Data
section are data lines until	len: equ $-msg ; "$" means "here"	1	Data
a non-data section begins.	; len is a value, not an address	0	
		0	
Start of a code/exsecutable	SECTION .text ; code section	1	Executable
section	global main ; make label available to linker	1	Executable
	main: ; standard gcc entry point	0	
		0	
	mov edx,len ; arg3, length of string to print	1	Executable
	mov ecx,msg ; arg2, pointer to string	1	Executable
	mov ebx,1 ; arg1, where to write, screen	1	Executable
	mov eax,4 ; write command to int 80 hex	1	Executable
	int 0x80 ; interrupt 80 hex, call kernel	1	Executable
		0	
	mov ebx,0 ; exit code, 0=normal	1	Executable
	mov eax,1 ; exit command to kernel	1	Executable
	int 0x80 ; interrupt 80 hex, call kernel	1	Executable