MCS Assembler

Specification

Date: 13.03.2014
(Revision 0.7)

Robert Bosch GmbH
Automotive Electronics (AE)

Copyright Notice and Proprietary Information

LEGAL NOTICE

The content of this document is subject to continuous developments and improvements. All particulars and its use contained in this document are given by BOSCH in good faith.

NO WARRANTIES: TO THE MAXIMUM EXTENT PERMITTED BY LAW, NEITHER THE INTELLECTUAL PROPERTY OWNERS, COPYRIGHT HOLDERS AND CONTRIBUTORS, NOR ANY PERSON, EITHER EXPRESSLY OR IMPLICITLY, WARRANTS ANY ASPECT OF THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, INCLUDING ANY OUTPUT OR RESULTS OF THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO UNLESS AGREED TO IN WRITING. THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO IS BEING PROVIDED "AS IS", WITHOUT ANY WARRANTY OF ANY TYPE OR NATURE, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND ANY WARRANTY THAT THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO IS FREE FROM DEFECTS.

ASSUMPTION OF RISK: THE RISK OF ANY AND ALL LOSS, DAMAGE, OR UNSATISFACTORY PERFORMANCE OF THIS SPECIFICATION (RESPECTIVELY THE PRODUCTS MAKING USE OF IT IN PART OR AS A WHOLE), SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO RESTS WITH YOU AS THE USER. TO THE MAXIMUM EXTENT PERMITTED BY LAW, NEITHER THE INTELLECTUAL PROPERTY OWNERS, COPYRIGHT HOLDERS AND CONTRIBUTORS, NOR ANY PERSON EITHER EXPRESSLY OR IMPLICITLY, MAKES ANY REPRESENTATION OR WARRANTY REGARDING THE APPROPRIATENESS OF THE USE, OUTPUT, OR RESULTS OF THE USE OF THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY, BEING CURRENT OR OTHERWISE. NOR DO THEY HAVE ANY OBLIGATION TO CORRECT ERRORS, MAKE CHANGES, SUPPORT THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, DISTRIBUTE UPDATES, OR PROVIDE NOTIFICATION OF ANY ERROR OR DEFECT, KNOWN OR UNKNOWN. IF YOU RELY UPON THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, YOU DO SO AT YOUR OWN RISK, AND YOU ASSUME THE RESPONSIBILITY FOR THE RESULTS. SHOULD THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL

LOSSES, INCLUDING, BUT NOT LIMITED TO, ANY NECESSARY SERVICING, REPAIR OR CORRECTION OF ANY PROPERTY INVOLVED TO THE MAXIMUM EXTEND PERMITTED BY LAW.

DISCLAIMER: IN NO EVENT, UNLESS REQUIRED BY LAW OR AGREED TO IN WRITING, SHALL THE INTELLECTUAL PROPERTY OWNERS, COPYRIGHT HOLDERS OR ANY PERSON BE LIABLE FOR ANY LOSS, EXPENSE OR DAMAGE, OF ANY TYPE OR NATURE ARISING OUT OF THE USE OF, OR INABILITY TO USE THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, INCLUDING, BUT NOT LIMITED TO, CLAIMS, SUITS OR CAUSES OF ACTION INVOLVING ALLEGED INFRINGEMENT OF COPYRIGHTS, PATENTS, TRADEMARKS, TRADE SECRETS, OR UNFAIR COMPETITION.

INDEMNIFICATION: TO THE MAXIMUM EXTEND PERMITTED BY LAW YOU AGREE TO INDEMNIFY AND HOLD HARMLESS THE INTELLECTUAL PROPERTY OWNERS, COPYRIGHT HOLDERS AND CONTRIBUTORS, AND EMPLOYEES, AND ANY PERSON FROM AND AGAINST ALL CLAIMS, LIABILITIES, LOSSES, CAUSES OF ACTION, DAMAGES, JUDGMENTS, AND EXPENSES, INCLUDING THE REASONABLE COST OF ATTORNEYS' FEES AND COURT COSTS, FOR INJURIES OR DAMAGES TO THE PERSON OR PROPERTY OF THIRD PARTIES, INCLUDING, WITHOUT LIMITATIONS, CONSEQUENTIAL, DIRECT AND INDIRECT DAMAGES AND ANY ECONOMIC LOSSES, THAT ARISE OUT OF OR IN CONNECTION WITH YOUR USE, MODIFICATION, OR DISTRIBUTION OF THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, ITS OUTPUT, OR ANY ACCOMPANYING DOCUMENTATION.

GOVERNING LAW: THE RELATIONSHIP BETWEEN YOU AND ROBERT BOSCH GMBH SHALL BE GOVERNED SOLELY BY THE LAWS OF THE FEDERAL REPUBLIC OF GERMANY. THE STIPULATIONS OF INTERNATIONAL CONVENTIONS REGARDING THE INTERNATIONAL SALE OF GOODS SHALL NOT BE APPLICABLE. THE EXCLUSIVE LEGAL VENUE SHALL BE DUESSELDORF, GERMANY.

MANDATORY LAW SHALL BE UNAFFECTED BY THE FOREGOING PARAGRAPHS.

INTELLECTUAL PROPERTY OWNERS/COPYRIGHT OWNERS/CONTRIBUTORS: ROBERT BOSCH GMBH, ROBERT BOSCH PLATZ 1, 70839 GERLINGEN, GERMANY AND ITS LICENSORS.

Revision History

Issue	Date	Remark
0.1	17.10.2011	Initial version
0.2	23.11.2011	Refined signature of instructions AWR, SHL, SHR in architecture MCS24-1. Fixed bug in generating definition files using command line option –odef: Symbol definitions that contain more than 20 characters caused program crash.
0.2.1	5.12.2011	Modified data type of generated C array and added extern statement to definition header file in C backend.
0.3	15.2.2012	Improved handling of negative numbers during assembling procedure: Negative numbers are only assembled if they can be fully represented within the range of the reserved bits in the instruction word. Otherwise an error is generated. Added VARIABLE directive to provide an architecture independent memory initialization possibility.
0.4	27.04.2012	Removed extern statement from header file generation in C backend. Removed path information in pre-processor variables generated in C-header file. Changed default value of –adrmax argument to 0x17FC. Added hash character (#) as second comment character. Added synonym for DEFINE directive (.set). Added new section defining a subset of assembler syntax that can be used for HIGH TECH assembler. Fixed bug in C-Backend: ASM-MCS crashed if the assembled result was 0 Bytes.
0.5	07.01.2013	Added new architecture MCS24-2 as mentioned in GTMSPEC2. Added new option –ofmt mif in order to generate memory initialization files in MIF format.

0.6	08.07.2013	Added new assembler directive CMDLINE. Allow blank as a separator after command line option –I. Added the following C-backend specific command line options: –otyp (defining output type with optional qualifier), –xtern (add extern statement to generated header file), and –hinc (add include statement for generated C-Header file at the beginning of the generated C file).
0.7	13.03.2014	Added new architecture MCS24-3 as mentioned in GTMSPEC3.

Tracking of major changes

Conventions

The following conventions are used within this document.

ARIAL BOLD CAPITALS	Names of signals
Arial bold	Names of files and directories
`Courier bold`	Command line entries
`Courier`	Extracts of files

References

This document refers to the following documents.

Ref	Authors(s)	Title
GTMSPEC1	AE/EIN2	GTM-IP Specification v1.5.5
GTMSPEC2	AE/EIN2	GTM-IP Specification v2.1.2.1
GTMSPEC3	AE/EIN2	GTM-IP Specification Appendix C v3.0.3.1
IFSDOC	AE/EIN2	IFS User's Guide March 2010
HIGHTEC	http://www.hightec-rt.com/	
ALTERA	http://www.altera.com/	

Terms and Abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
IFS	Interface Simulator

Table of Contents

1 Introduction 1
1.1 Overview 1
2 Assembler File Syntax 3
2.1 General 3
2.2 Instructions 3
2.3 Directives 4
2.3.1 INCLUDE directive 4
2.3.2 ORG directive 4
2.3.3 DEFINE directive 5
2.3.4 LABEL directive 5
2.3.5 REGISTER directive 6
2.3.6 VARIBLE directive 6
2.3.7 CMDLINE directive 7
2.4 Arithmetic Expressions 7
2.5 Example 8
2.6 Third Party Tools Compatibility 10
3 Machine Code generation 11
3.1 C Code Generation 11
3.2 IFS Code Generation 13
3.3 LOG File Generation 14
4 Command Line Options 16
5 Target Architectures 19
5.1 MCS24-1 Architecture 19
5.1.1 Types 19
5.1.2 Instructions 20
5.1.3 Memory Initialization 21
5.2 MCS24-2 Architecture 21
5.2.1 Types 21
5.2.2 Instructions 22
5.3 MCS24-3 Architecture 24
5.3.1 Types 24
5.3.2 Instructions 25

1 Introduction

1.1 Overview

This document describes the features and the usage of ASM-MCS, an assembler tool for generating machine code for the Multichannel Sequencer (MCS). The MCS is a programmable RISC like sequencer module, which executes several parallel working tasks on a single processing core. Details about the MCS can be found in [GTMSPEC1, GTMSPEC2, GTMSPEC3].

The behaviour of the different MCS tasks for a single MCS module can be specified in one MCS assembler source files, referred as MCS file in the following, which typically has the file extension ".mcs". ASM-MCS translates this file into machine code that is loaded into the dedicated MCS RAM module. The generated machine code can be represented in the following formats:

- Initialized C-Code Array, that can be processed by a C-Compiler,
- IFS-Command-File with memory initialization commands than can be processed by any SystemC-IFS based simulation environment.

Details about the SystemC-IFS based simulation environment can be found in [IFSDOC]. If the assembler source code for the different MCS tasks should be distributed among different MCS files, the different files can be included by a top level assembler file.

2 Assembler File Syntax

2.1 General

An MCS file typically consists of a list of MCS instructions mixed up with several assembler directives, which are not directly translated into machine code, but they are used to tell the tool ASM-MCS how to translate the instructions. Each non-empty line of an MCS-File may either contain exactly one MCS instruction or one assembler directive. Moreover, ASM-MCS supports line comments everywhere in the MCS-file, whereas a line comment is introduced by the character semicolon (';') or the character hash ('#') .

2.2 Instructions

Each instruction of an MCS file corresponds to the following form:

```
[<LABEL> : ]   <MNEMONIC> [<ARG1> [,] <ARG2> … [,] <ARGN>],
```

whereas <MNEMONIC> is a string that identifies the instruction to be coded. The string <MNEMONIC> is not case sensitive. An instruction may also be qualified by an optional and unique string identifier <LABEL>, which can be referred in the source code as a so called label that holds the actual address of the instruction. These labels are typically used by branch instructions to code the desired program flow, or by memory access instructions to identify the address of memory mapped program variables. It should be noted that ASM-MCS handles the identifier <LABEL> not in a case sensitive manner. The string identifiers of labels must be accepted by the following regular expression: [a-zA-Z][_a-zA-Z0-9]*. Some instructions also require a list of arguments <ARG1> to <ARGN>. The arguments may optionally be separated by a coma (','). The expected type and the number of expected arguments depend on the actual instruction. ASM-MCS distinguishes between the following types of arguments:

- **Register Argument:** A register argument identifies a general purpose register or a special function register by a predefined register symbol (e.g. R7 or STA), whereas each symbol maps to a unique number. There exist different types of predefined register symbols and different arguments will support different types of registers. Details about the instructions and its accepted register types can be found in section 5. A second possibility for specifying register arguments is the usage of user defined register identifiers, whereas a desired string identifier is mapped to a number. User defined register identifiers are generated by the directive REGISTER, as explained in section 2.3.5. ASM-MCS will check if the value of a user defined register can be mapped to the

register type of the instruction's actual argument. Register arguments do not accept arithmetic expression. In this case ASM-MCS will report an error message.

- **Literal Argument:** Literal arguments accept arbitrary arithmetic expressions consisting of constant numbers, variables, labels, and arithmetic operators. Details about arithmetic expressions can be found in section 2.3.5. Literal arguments are mainly used for coding immediate operands in instructions or address operands in branch and memory access instructions. ASM-MCS reports an error, if the result of an arithmetic expression exceeds the valid range of a literal argument. Literal Arguments do not accept symbols of register types or symbols of user defined registers.

2.3 Directives

ASM-MCS provides a set of so called assembler directives that are controlling the assembling procedure. If nothing other is outlined in this documentation, the specification of the assembler directives within the MCS file is case sensitive. The following subsections describe the available assembler directives.

2.3.1 INCLUDE directive

The INCLUDE directive can be used to insert the content of another MCS file in the current MCS file. It can be used for structuring the MCS code in order to improve readability and reusability. The syntax of the INCLUDE directive is as follows:

```
.include "<INCFILE>"
```

The string <INCFILE> identifies the name of the file that will be included at the current position. <INCFILE> can either be a single file name or a file name with absolute or relative path information. ASM-MCS tries to open the specified file within the current working directory. If the file cannot be found, it will evaluate a list of include directories that can be specified on command line. Details about the command line options can be found in section 4.

2.3.2 ORG directive

The ORG directive can be used to manipulate the address counter of ASM-MCS in order to control the memory regions in which ASM-MCS will put the assembled code. Moreover, the ORG directive can be used in conjunction with the LABEL directive (see section 2.3.4) to reserve data sections in the memory to store program variables. The syntax of the ORG directive is as follows:

```
.org <EXPR>
```

The argument <EXPR> can be any arithmetic expression, consisting of numeric constants, variables, labels and arithmetic operators. Details about arithmetic expression can be found in section 2.4. The ORG directive relocates the current address counter to the value of <EXPR>, which means that the assembled data following the ORG directive will be located after address <EXPR> in the memory. It should be noted that the ORG directive only accepts values that are integer multiples of 4. Backward relocation of the address counter is not supported and will generate an error message.

2.3.3 DEFINE directive

The DEFINE directive can be used to create assembler variables within ASM-MCS that are present during the assembling procedure. Assembler variables can be used within any arithmetic expressions that are following the first definition of a variable. Assembler variables can be used in the MCS file to write generic assembler code. Moreover, the usage of variables improves readability of the code. The syntax of the DEFINE directive is as follows:

```
.define <VARNAME> <EXPR>
```

A DEFINE directive generates a variable with a symbol name <VARNAME> and it initializes the variable with the value of the arithmetic expression <EXPR>. It should be noted that the identifier <VARNAME> is must be accepted by the following regular expression: `[a-zA-Z][_a-zA-Z0-9]*` and the variable handling of ASM-MCS is not case sensitive. The variables of ASM-MCS are always 32-bit wide unsigned numbers. A DEFINE directive may overwrite variables that were previously created with a DEFINE directive. However, predefined or user defined register symbols, as well as labels cannot be overwritten with a DEFINE directive.

The following syntax for the DEFINE directive is equivalent to the syntax mentioned above:

```
.set <VARNAME> [,] <EXPR>
```

2.3.4 LABEL directive

A LABEL directive can be used to create symbolic identifiers and map the current value of the address counter to this symbol. Such kind of identifier is called label. The syntax for creating a label with the symbolic name <LABEL> is as follows:

```
<LABEL> :
```

The usage of the LABEL directive is equal to the usage of an instruction with a qualified label, as mentioned in section 2.2. However, the LABEL directive can be placed in arbitrary contexts (e.g. between two ORG directives). Labels are stored as 32-bit wide values. The usage of labels and variables is identical in the most cases. However, there are a few differences between labels and variables: First, a label cannot overwrite already defined symbols (e.g defined by a DEFINE directive, a REGISTER directive, or another LABEL directive). The second difference is the fact that labels can be used in arithmetic expressions of instructions' arguments without a previous declaration of the label. However the definition of the referred label must occur in following statements of the source code. This is mechanism only possible for labels used in the context of instruction arguments. It is required to enable the coding of forward branch statements.

2.3.5 REGISTER directive

A REGISTER directive enables the creation of symbolic names for registers. Although ASM-MCS defines intrinsically symbolic names for all available registers it may be helpful to use application specific names as register arguments. The syntax for the REGISTER directive is as follows:

```
.register <REGNAME> <EXPR>
```

The REGISTER directive generates a symbolic name `<REGNAME>` of a user defined register and maps the value of the arithmetic expression `<EXPR>` to that name. It should be noted that the identifier `<REGNAME>` is must be accepted by the following regular expression: `[a-zA-Z][_a-zA-Z0-9]*` and the user defined register handling of ASM-MCS is not case sensitive. The user defined registers of ASM-MCS are always 32-bit wide unsigned numbers. User defined registers can only be applied to register arguments in instructions. They cannot be used in arithmetic expressions of directives or literal arguments of instructions. If a user defined register is used as register argument in an instruction, ASM-MCS checks if the assigned value of the user defined register can be mapped to the expected register type of the instruction's argument. A REGISTER directive cannot overwrite any other symbols (e.g. labels, user defined register, predefined registers or variables).

2.3.6 VARIBLE directive

The VARRIABLE directive provides a possibility to initialize program variables that are located in the MCS memory. The syntax for the VARIABLE directive is as follows:

```
.var <EXPR> [<WIDTH>]
```

Calling the VARIABLE directive without the optional argument <WIDTH> initializes a memory variable with the value <EXPR> using the underlying word width of the currently selected MCS architecture. ASM-MCS will generate an error message if <EXPR> can not be represented within the architecture's word width. If the optional argument <WIDTH> is specified, ASM-MCS will use this value as the word width for memory initialization. <WIDTH> must be in the range from 1 to the word width of the architecture's instruction width.

2.3.7 CMDLINE directive

The CMDLINE directive provides a possibility to add command line options directly within the assembler source file. The syntax for the CMDLINE directive is as follows:

```
.cmdline <OPTION>*
```

whereas the optional option list <OPTION> may contain arbitrary command line options in arbitrary order. The available command line options are mentioned in section 4. Whenever <OPTION> contains a command line option that has already been defined directly during the call of ASM-MCS it will be overridden be the value given in the CMDLINE directive. If the value of a command line option contains any spaces (e.g. in the name of a directory or file) it must be specified in quotes (“”) as the following example shows:

```
.cmdline –I ”c:\My Dir” –ofmt c –o myfile.c –odef myfile.h
```

It should be noted that the CMDLINE directive is only accepted as a first statement in the assembler source file.

2.4 Arithmetic Expressions

ASM-MCS supports arithmetic expressions as arguments of assembler directives or as literal arguments of instructions. An arithmetic expression consists of arbitrary conjunctions of constant numbers, variables, or labels. All arithmetic expressions are applied with an underlying 32-bit wide arithmetic of unsigned numbers. The following operators are available:

|	bitwise inclusive OR conjunction
^	bitwise exclusive OR conjunction
&	bitwise AND conjunction
+	addition operator
-	subtraction operator (binary operator)
*	multiplication operator

/	divison operator
%	modulo operator
**	power operator
~	bitwise NOT operator (unary operator)
-	negation operator (unary operator)

If nested operators are used in arithmetic expressions, the operator precedence is applied in ascending order concerning the list of operators mentioned above. Arithmetic expressions can use round brackets (‘(’ and ‘)’) to create terms with arbitrary operator precedence. Labels and variables are referred in arithmetic expressions just by writing the names. Constant numbers can be specified as decimal numbers, hexadecimal numbers or binary numbers. The following regular expressions declare the all the possibilities for constant number specifications:

[0-9]+	decimal number
d#[0-9]+	decimal number
0x[0-9A-Fa-f]+	hexadecimal number
h#[0-9A-Fa-f]+	hexadecimal number
$[0-9A-Fa-f]+	hexadecimal number
b#[0-1]+	binary number

The following simple example demonstrates how arithmetic expressions can be used in the tool ASM-MCS:

```
.define X    3
.define Y    0xFF
.define Z    (2**X+2)/2+Y
```

The tool will evaluate the variables as X = 3, Y = 255 and Z = 260 during the assembling procedure.

2.5 Example

Figure 2.1 shows a part of a more complex MCS file in order to demonstrate the syntax of ASM-MCS. The first statement `.include “mcs24_1.inc”` of this example includes the file “mcs24_1.inc” that contains common definitions for the architecture `mcs24-1`. This file is part of the ASM-MCS deliverable and it is recommended to include that file, whenever the architecture `mcs24-1` is targeted. In the next section of the MCS file a variable `EN_L_MSK` is defined and initialized with a hexadecimal value. In section 3 of the source file, the reset vectors of the used MCS channels are initialized. The statement `.org 0x0` causes the assembler to begin assembling at the first memory location. The two statements following the statement `.org 0x0` are unconditional jump statements that branch to the program entry points of MCS

```
; 1) include architecture specific definitions
; ---------------------------------------------
.include “mcs24_1.inc”

; 2) define some variables
; -------------------------
.define EN_L_MSK        $FFFFFE

; 3) initialize reset vectors of MCS channels 0 and 1
; ----------------------------------------------------
.org 0x0
jmp tsk0_init
jmp tsk1_init

; 4) allocate and initialize memory variables
; -------------------------------------------
.org h#20
var_a: .var 17 ; initialize variable var_a with value of 17

; 5) allocate stack frames (each task has 16 memory locations)
; ------------------------------------------------------------
.org $40
tsk0_stack:
.org (tsk0_stack+0x40)
tsk1_stack:
.org (tsk1_stack+0x40)

# 6) program entry for MCS-channel 0
# ----------------------------------
tsk0_init:
      movl  R7  (tsk0_stack-4) # initialize stack pointer of task 0
      mrd   R0  var_a          # load var_a to register R0
      call  delay_loop         # run delay loop subprogram
      andl  STA EN_L_MSK       # disable MCS channel 0

; 7) procedure delay loop
; -----------------------
delay_loop:
      subl R0  1              ; decrement R0
      jbc  STA Z delay_loop   ; iterate loop while zero flag is cleared
      ret                     ; return to caller

; 8) program entry for MCS-channel 1
; ----------------------------------
tsk1_init:
      movl  R7  (tsk1_stack-4)  ; initialize stack pointer of task 1
      …
```

Figure 2.1: Example of an MCS file.

channel 0 and MCS channel 1. These program entry points are referred by the two labels `tsk0_init` and `tsk1_init`. Section 4 shows how to allocate and initialize a 24 bit wide program variable var_a = 17 in the MCS memory using a label var_a and a VARIABLE directive. Details about the available instructions can be found in section 5 of this document. Since the `.var` statement is following immediately the statement `.org 0x20`, the variable var_a is placed to the hexadecimal memory address 0x20. Section 5 of the example shows how to use the `.org` statement and the LABEL directive to allocate a stack frame for each MCS channel with 16 memory entries (size of 0x40 bytes). Stack frame `tsk0_stack` ranges from 0x40 to 0x7C and stack frame `tsk1_stack` ranges from 0x80 to 0xBC. Section 6 of the example shows the program entry point of MCS-channel 0. It begins at address 0xC0. After initialization of the stack pointer register R7 with the beginning of the allocated stack frame `tsk0_stack`, MCS-channel zero reads the memory variable var_a to register R0 and it calls the sub routine delay_loop described in section 7 of the example. This subroutine embeds a delay loop that iterates var_a-times (17-times) and then it returns to the caller. The initialization of the stack pointer register R7 also shows how to use ASM-MCS for evaluation of arithmetic expressions within the literal argument of the MOVL instruction.

2.6 Third Party Tools Compatibility

This section defines a subset of the introduced assembler syntax that guarantees code compatibility between ASM-MCS and other MCS related assembler tools, provided by third party tool vendors (e.g. [HIGHTEC]). The compatible subset is defined as wollows:

1. Line comments must be introduced by the character hash (‘#’).
2. Arguments of instructions must be separated by a comma (‘,’).
3. Hexadecimal expressions must be specified with the prefix (“0x”).
4. Decimal expressions must be specified without a prefix.
5. Binary expressions must not be specified.
6. The following assembler directives must not be specified: REGISTER directive.
7. The following arithmetic operators must not be specified: power operator (“**”), division operator (‘/’), module operator (‘%’).
8. The VARIABLE directive must be specified using the syntax with the set-keyword (“.set”).

3 Machine Code generation

As already mentioned, ASM-MCS provides the generated machine code in form of C code as well as IFS code that can be used for system integration or simulation. Moreover, ASM-MCS can also generate a memory initialization file in MIF format containing the MCS machine code that can be used with third party tools (e.g. Altera Quartus [ALTERA]) for initialization of FPGA memories. Besides these formats, ASM-MCS can also generate an additional LOG file for debugging purposes. The LOG-File contains both, MCS source files and generated machine code. The generation of the desired output formats can be controlled by the command line options of ASM-MCS as described in section 4.

3.1 C Code Generation

ASM-MCS can generate a C file containing a C array with the generated machine code. Moreover, ASM-MCS can also generate an optional C header file, called definition file that contains additional symbolic definitions about the labels used in the MCS File. Figure 3.1 shows an example of a generated C file and Figure 3.2 the corresponding C header definition File. The files were generated using the MCS file in the example of Figure 2.1. Figure 3.1 shows that the generated C array has the name mcs0_mem and it is of type unsigned.

```
/* generated by MCS-Assembler tool ASM-MCS version 0.7 */
/* Copyright (C) 2011-2013 by Robert Bosch GmbH, Germany */
/* target architecture : mcs24-1 */

unsigned long mcs0_mem[56] = {
    0xE00000C0,
    0xE00000DC,
    0x00000000,
    0x00000000,
    0x00000000,
    0x00000000,
    0x00000000,
    0x00000000,
    0x00000011,
    0x00000000,
…
    0x1700003C,
    0xA0010020,
    0xE00300D0,
    0x48FFFFFE,
    0x30000001,
    0xE85200D0,
    0xE0040000,
    0x1700007C
};
```

Figure 3.1: Example of a generated C file with machine code.

The name of the generated array can be redefined on the command line. Each non-zero element of the C array corresponds to an assembled instruction or an initialized memory mapped variable. The size of the generated array always ranges from the first to the last memory location that has been inspected by ASM-MCS. This means that the first memory location may not to correspond to the first memory location of the associated MCS RAM module. For example, if the first statement in the MCS-File of would be .org 0x10, the first array element would correspond to the MCS memory address 0x10. The corresponding offset between the first MCS memory address and the first element of the C array is defined by the C pre-processor variable OFFSET_MCS0_MEM in the C header file, as shown in Figure 3.2. The variable SIZE_MCS0_MEM holds the total size in bytes for the generated C array mcs0_mem. The C header file also provides a pre-processor variable definition for each label mentioned in the MCS file. If the C array is directly mapped to the MCS memory, the software of the host CPU can use these pre-processor variables for direct access of MCS program variables. For example, the host CPU could overwrite the value 17 of variable var_a in the example of Figure 2.1 with the value 19 using the following C code statement:

```
mcs0_mem[LABEL_MCS0_MEM_VAR_A] = 19;
```

```
/* generated by MCS-Assembler tool ASM-MCS version 0.7 */
/* Copyright (C) 2011-2013 by Robert Bosch GmbH, Germany */
/* target architecture : mcs24-1 */

#ifndef EXAMPLE_H_
#define EXAMPLE_H_

#define OFFSET_MCS0_MEM     (   0) /* byte address offset */
#define SIZE_MCS0_MEM       ( 224) /* code size in bytes */

#define LABEL_MCS0_MEM_DELAY_LOOP               (  52)
#define LABEL_MCS0_MEM_TSK0_INIT                (  48)
#define LABEL_MCS0_MEM_TSK1_STACK               (  32)
#define LABEL_MCS0_MEM_TSK1_INIT                (  55)
#define LABEL_MCS0_MEM_TSK0_STACK               (  16)
#define LABEL_MCS0_MEM_VAR_A                    (   8)

extern unsigned long mcs0_mem[];

#endif
```

Figure 3.2: Example of a generated C header file with symbolic definitions.

3.2 IFS Code Generation

ASM-MCS can also generate an IFS command file code in order to simulate MCS programs in a SystemC-IFS based simulation environment. Details about SystemC-IFS and its script language can be found in [IFSDOC]. According to the C code generation, ASM-MCS generates an IFS command file for memory initialization and an optional definition file that contains symbolic name definitions about the labels used in the MCS file. Figure 3.3 shows an example of a generated IFS command file and Figure 3.4 the corresponding command file with symbol definitions. The files were generated using the MCS-File in the example of Figure 2.1. Figure 3.3 shows that the generated IFS code only overwrites the memory locations, where the ASM-MCS generated any code for instructions or initialized variables. All other memory locations are not touched by this IFS code. The generated name of the MCS memory identifier (here mcs0_mem) can be redefined on the command line of ASM-MCS.

```
-- generated by MCS-Assembler tool ASM-MCS version 0.7
-- Copyright (C) 2011-2013 by Robert Bosch GmbH, Germany
-- target architecture : mcs24-1

AEI WRITE $(#mcs0_mem+h#00000010)      h#E00000C0
AEI WRITE $(#mcs0_mem+h#00000014)      h#E00000DC
AEI WRITE $(#mcs0_mem+h#00000020)      h#00000011
AEI WRITE $(#mcs0_mem+h#000000C0)      h#1700003C
AEI WRITE $(#mcs0_mem+h#000000C4)      h#A0010020
AEI WRITE $(#mcs0_mem+h#000000C8)      h#E00300D0
AEI WRITE $(#mcs0_mem+h#000000CC)      h#48FFFFFE
AEI WRITE $(#mcs0_mem+h#000000D0)      h#30000001
AEI WRITE $(#mcs0_mem+h#000000D4)      h#E85200D0
AEI WRITE $(#mcs0_mem+h#000000D8)      h#E0040000
AEI WRITE $(#mcs0_mem+h#000000DC)      h#1700007C
```

Figure 3.3: Example of a generated IFS command file with machine code.

The IFS variable OFFSET_MCS0_MEM in Figure 3.4 holds the offset address in bytes from the beginning of the MCS memory to the first memory location, which is modified by the generated IFS code. IFS variable SIZE_MCS0_MEM refers the actual MCS memory size in bytes, which is addressed by ASM-MCS. In other words, the generated code of ASM-MCS is ranges from address OFFSET_MCS0_MEM to OFFSET_MCS0_MEM + SIZE_MCS0_MEM. Moreover ASM-MCS creates for each label given in the MCS file an additional IFS variable in the definition file of Figure 3.4. These labels can be used to access MCS variables directly in the IFS command file code. For example, the following statement overwrites the value 17 of variable var_a in the example of. Figure 2.1 with the value 19:

```
AEI WRITE #LABEL_MCS0_MEM_VAR_A    19
```

```
-- generated by MCS-Assembler tool ASM-MCS version 0.7
-- Copyright (C) 2011-2013 by Robert Bosch GmbH, Germany
-- target architecture : mcs24-1

AEI ASSIGN OFFSET_MCS0_MEM  =   16  -- byte address offset
AEI ASSIGN SIZE_MCS0_MEM    =  208  -- code size in bytes

AEI ASSIGN LABEL_MCS0_MEM_DELAY_LOOP      = $(#mcs0_mem+h#000000D0)
AEI ASSIGN LABEL_MCS0_MEM_TSK0_INIT       = $(#mcs0_mem+h#000000C0)
AEI ASSIGN LABEL_MCS0_MEM_TSK1_STACK      = $(#mcs0_mem+h#00000080)
AEI ASSIGN LABEL_MCS0_MEM_TSK1_INIT       = $(#mcs0_mem+h#000000DC)
AEI ASSIGN LABEL_MCS0_MEM_TSK0_STACK      = $(#mcs0_mem+h#00000040)
AEI ASSIGN LABEL_MCS0_MEM_VAR_A           = $(#mcs0_mem+h#00000020)
```

Figure 3.4: Example of a generated IFS command file with symbolic definitions.

3.3 LOG File Generation

In order to inspect the assembling procedure of ASM-MCS, it is possible to generate a LOG file for an assembling procedure. This LOG file contains both, the source code as well as the machine code that is generated. Figure 3.5 shows the LOG file that is generated by ASM-MCS during assembling the MCS file of Figure 2.1. The right hand side of the LOG file contains the original source code including the line numbers and the left hand side of the LOG file shows the generated machine in the form [<ADDR>] = <CODE>, whereas <ADDR> is the memory address to which ASM-MCS puts the assembled code <CODE>. The generated machine code on the left hand side of a specific line always results by translating the mnemonic instruction on the right hand side in the same line. If the MCS file includes other files using the INCLUDE directive, the referred file will be extracted in the LOG file. Moreover, the LOG file also contains a symbol table, which lists all available symbols and its values after the assembling procedure.

```
                              |     4:
                              |     5: ; 2) define some variables
                              |     6: ; -------------------------
                              |     7: .define EN_L_MSK       $FFFFFE
                              |     8:
                              |     9: ; 3) initialize reset vectors of MCS
                              |    10: ; ----------------------------------
                              |    11: .org 0x10
[0x00000010] = 0xE00000C0     |    12: jmp tsk0_init
[0x00000014] = 0xE00000DC     |    13: jmp tsk1_init
                              |    14:
                              |    15: ; 4) allocate and initialize memory v
                              |    16: ; ----------------------------------
                              |    17: .org h#20
[0x00000020] = 0x00000011     |    18: var_a: .var 17 ; initialize variable
                              |    19:
                              |    20: ; 5) allocate stack frames (each task
                              |    21: ; ----------------------------------
                              |    22: .org $40
                              |    23: tsk0_stack:
                              |    24: .org (tsk0_stack+0x40)
                              |    25: tsk1_stack:
                              |    26: .org (tsk1_stack+0x40)
                              |    27:
                              |    28: # 6) program entry for MCS-channel 0
                              |    29: # ---------------------------------
                              |    30: tsk0_init:
[0x000000C0] = 0x1700003C     |    31:      movl  R7  (tsk0_stack-4)# initia
[0x000000C4] = 0xA0010020     |    32:      mrd   R0  var_a         # load
[0x000000C8] = 0xE00300D0     |    33:      call  delay_loop        # run de
[0x000000CC] = 0x48FFFFFE     |    34:      andl  STA EN_L_MSK      # disabl
                              |    35:
                              |    36: ; 7) procedure delay loop
                              |    37: ; ------------------------
                              |    38: delay_loop:
[0x000000D0] = 0x30000001     |    39:      subl R0  1              ; decrem
[0x000000D4] = 0xE85200D0     |    40:      jbc  STA Z delay_loop   ; iterat
[0x000000D8] = 0xE0040000     |    41:      ret                     ; return
                              |    42:
                              |    43: ; 8) program entry for MCS-channel 1
                              |    44: ; ---------------------------------
                              |    45: tsk1_init:
[0x000000DC] = 0x1700007C     |    46:      movl  R7  (tsk1_stack-4)  ; init
                              |    47:
```

Figure 3.5: Example of a LOG file containing source code and machine code.

4 Command Line Options

This section describes the available command line options of ASM-MCS. The command line is executed as

```
asm-mcs <OPTIONS>* <ASMFILE>,
```

whereas <ASMFILE> refers the file name of the MCS assembler source file. <ASMFILE> may include absolute or relative path information. The optional option list <OPTIONS> may contain the following items in arbitrary order:

- `-arch <ARCH>`

  This option selects the desired target architecture for the MCS. Details about the available target architectures and the corresponding values for its identifier <ARCH> can be found in section 5. The default value for <ARCH> is `mcs24-1`.

- `-ofmt <OFMT>`

  This option defines the de desired output format. If <OFMT> contains the value `c` a C file with an array definition is generated, as explained previously. This is also the default value for this option. If <OFMT> has the value `ifs` an IFS command file will be generated instead. If <OFMT> has the value `mif` a Memory initialization file in MIF format is created as output file. This option cannot be used with in conjunction with option `-odef`.

- `-olbl <LABEL>`

  This option defines the symbolic name <LABEL> for the memory label that is used in the generated files. In the case of the C code generation <LABEL> defines the name of the C array and in the case of the IFS generation <LABEL> is the name of IFS variable that refers the requested MCS memory. The default value for label is `mcs0_mem`.

- `-odef <DFILE>`

  This option enables the generation of an additional definition file with the name <DFILE>. If C-code generation is requested, a C header file according to section 3.1 is generated. In the case of IFS code generation and IFS command file with symbol definition according to section 3.2 is generated.

- `-o <OFILE>`

  This option enables the definition of an alternative name <OFILE> for the generated output file. The default name is the name of the input file (without the extension .mcs) plus the extension .c, in the case of C code generation, or the extension .ifs in the case of an IFS code generation, or the extension .mif in the case of a MIF file generation.

- `-log <LFILE>`

  This option enables the generation of an additional LOG file definition file with the name <LFILE>.

- `-adrmax <ADDR>`

  This option performs an additional range check of the maximum available address <ADDR> during the assembling procedure. The default value for <ADDR> is 0x17FC.

- `-I<INCDIR>` or `-I <INCDIR>`
  This option appends the directory <INCDIR> to the list of available include directories, in which ASM-MCS tries to open files referred with the INCLUDE directive. This option may be announced several times on the command line.

- `-W`

  This option prints all warnings at the end of a successful assembling. In the default case, warnings are only printed when additional errors occurred.

- `--version`

  This option prints version information on the console.

- `-h` or `--help`

  This option prints a list with all available command line options.

- `-otyp <TYPE>`

This option defines an output type with optional type qualifiers for the array definition of the generated C file. A meaningful value for <TYPE> could be “const unsigned int”. The default value for <TYPE> is “unsigned long”. It should be noted that ASM-MCS does not check if <TYPE> is a valid C type with optional qualifiers. The option `–otyp` is only applicable if option `-ofmt` is set to `c`.

- `-hinc`

This option adds an additional include pre-processor statement to the beginning of a generated C file in order to include the generated header file defined by command line option `-odef`. The option `–hinc` is only applicable if option `-ofmt` is set to `c` and option `-odef` is used.

- `-xtern`

This option adds an additional extern statement into the generated C header file in order to declare the generated array definition as an externally defined variable. This option is only applicable if the option `-ofmt` is set to `c` and the option `-odef` is used.

5 Target Architectures

This section describes the available MCS target architectures of ASM-MCS. The required MCS target architecture has to be selected by the architecture command line switch

```
-arch <ARCH>.
```

5.1 MCS24-1 Architecture

This MCS target architecture is selected by using the value

```
<ARCH> = mcs24-1
```

for the architecture command line switch `-arch`. ASM-MCS will assume an MCS architecture specification according to [GTMSPEC1].

5.1.1 Types

ASM-MCS provides register argument types defining a set of available symbols that can be used as register arguments. The following types are available in this architecture

```
<AREG>    ::= R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 | ZERO

<OREG>    ::= R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 | STA | ACB
            | CTRG | STRG | TBU_TS0| TBU_TS1 | TBU_TS2 | MHB
```

In order to specify literal arguments of instructions and memory initializations, the following literal types are available:

```
<LIT32>   ::= A 32 bit literal value in the range [0; 2^32-1]

<LIT24>   ::= A 24 bit literal value in the range [0; 2^24-1]

<LIT16>   ::= A 16 bit literal value in the range [0; 2^16-1]

<LIT9>    ::= A 9 bit literal value in the range [0;511]

<LIT4>    ::= A 4 bit literal value in the range [0;15]
```

```
<RNG0TO23> ::= Literal value in the range [0;23]

<RNG0TO24> ::= Literal value in the range [0;24]
```

5.1.2 Instructions

In this architecture ASM-MCS implements the instructions set as mentioned in [GTMSPEC1]. ASM-MCS uses the following rules in a non case sensitive manner to match an instruction:

```
<INSTR>::=     NOP
     |         MOVL    <OREG> <LIT24>
     |         ADDL    <OREG> <LIT24>
     |         SUBL    <OREG> <LIT24>
     |         ANDL    <OREG> <LIT24>
     |         ORL     <OREG> <LIT24>
     |         XORL    <OREG> <LIT24>
     |         ATUL    <OREG> <LIT24>
     |         ATSL    <OREG> <LIT24>
     |         BTL     <OREG> <LIT24>
     |         MOV     <OREG> <OREG>
     |         MRD     <OREG> <LIT16>
     |         MWR     <OREG> <LIT16>
     |         MRDI    <OREG> <OREG>
     |         MWRI    <OREG> <OREG>
     |         POP     <OREG>
     |         PUSH    <OREG>
     |         MWR24   <OREG> <LIT16>
     |         MWRI24  <OREG> <OREG>
     |         ARD     <AREG> <AREG> <LIT9>
     |         AWR     <OREG> <OREG> <RNG0TO23>
     |         ARDI    <AREG> <AREG>
     |         AWRI    <OREG> <OREG>
     |         NARD    <AREG> <AREG> <LIT9>
     |         NARDI   <AREG> <AREG>
     |         ADD     <OREG> <OREG>
     |         SUB     <OREG> <OREG>
     |         NEG     <OREG> <OREG>
     |         AND     <OREG> <OREG>
     |         OR      <OREG> <OREG>
     |         XOR     <OREG> <OREG>
```

```
|          SHR     <OREG> <RNG0TO24>
|          SHL     <OREG> <RNG0TO24>
|          ATU     <OREG> <OREG>
|          ATS     <OREG> <OREG>
|          BT      <OREG> <OREG>
|          JMP     <LIT16>
|          JBS     <OREG> <LIT4> <LIT16>
|          JBC     <OREG> <LIT4> <LIT16>
|          CALL    <LIT16>
|          RET
|          WURM    <OREG> <OREG> <LIT16>
```

5.1.3 Memory Initialization

The following obsolete memory initialization statements are available in this architecture:

```
<MEMINIT>::=   LIT24    <LIT24>
          |    LIT32    <LIT32>
```

Memory initialization of variables should be applied using the VARIABLE directive as mentioned in section 2.3.6 .

5.2 MCS24-2 Architecture

This MCS target architecture is selected by using the value

```
<ARCH> = mcs24-2
```

for the architecture command line switch `-arch`. ASM-MCS will assume an MCS architecture specification according to [GTMSPEC2].

5.2.1 Types

ASM-MCS provides register argument types defining a set of available symbols that can be used as register arguments. The following types are available in this architecture

```
<AREG>     ::= R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 | ZERO

<OREG>     ::= R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 | STA | ACB
             | CTRG | STRG | TBU_TS0| TBU_TS1 | MHB
```

In order to specify literal arguments of instructions and memory initializations, the following literal types are available:

```
<WLIT>    ::= A 24 bit word literal value
              in the range [0; 2^24-1]

<ALIT>    ::= A 10 bit address literal value
              in the range [0; 2^10-1]

<AOLIT>   ::= A 10 bit address offset literal value
              in the range [-2^9-1; 2^9-1]

<ARDLIT>  ::= A 9 bit ARU read address literal value
              in the range [0; 2^8-1]

<AWRLIT>  ::= A 5  bit ARU write index literal value
              in the range [0; 23]

<SFTLIT>  ::= A 5 bit shift literal value in the range [0; 24]

<BITLIT>  ::= A 4 bit shift literal value in the range [0; 15]

<MSKLIT>  ::= A 16 bit mask literal value
              in the range [0; 2^16-1]
```

5.2.2 Instructions

In this architecture ASM-MCS implements the instructions set as mentioned in [GTMSPEC1]. ASM-MCS uses the following rules in a non case sensitive manner to match an instruction:

```
<INSTR>::=     NOP
     |         MOVL    <OREG> <WLIT>
     |         ADDL    <OREG> <WLIT>
     |         SUBL    <OREG> <WLIT>
     |         ANDL    <OREG> <WLIT>
     |         ORL     <OREG> <WLIT>
     |         XORL    <OREG> <WLIT>
     |         ATUL    <OREG> <WLIT>
     |         ATSL    <OREG> <WLIT>
     |         BTL     <OREG> <WLIT>
```

```
|           MOV     <OREG> <OREG>
|           MRD     <OREG> <ALIT>
|           MWR     <OREG> <ALIT>
|           MRDI    <OREG> <OREG> [<AOLIT>]
|           MWRI    <OREG> <OREG> [<AOLIT>]
|           POP     <OREG>
|           PUSH    <OREG>
|           MWRL    <OREG> <ALIT>
|           MWRIL   <OREG> <OREG>
|           ARD     <AREG> <AREG> <ARDLIT>
|           AWR     <OREG> <OREG> <AWRLIT>
|           ARDI    <AREG> <AREG>
|           AWRI    <OREG> <OREG>
|           NARD    <AREG> <AREG> <ARDLIT>
|           NARDI   <AREG> <AREG>
|           ADD     <OREG> <OREG>
|           SUB     <OREG> <OREG>
|           NEG     <OREG> <OREG>
|           AND     <OREG> <OREG>
|           OR      <OREG> <OREG>
|           XOR     <OREG> <OREG>
|           SHR     <OREG> <SFTLIT>
|           SHL     <OREG> <SFTLIT>
|           ASRU    <OREG> <OREG>
|           ASRS    <OREG> <OREG>
|           ASL     <OREG> <OREG>
|           MINU    <OREG> <OREG>
|           MINS    <OREG> <OREG>
|           MAXU    <OREG> <OREG>
|           MAXS    <OREG> <OREG>
|           ATU     <OREG> <OREG>
|           ATS     <OREG> <OREG>
|           BT      <OREG> <OREG>
|           JMP     <ALIT>
|           JBS     <OREG> <BITLIT> <ALIT>
|           JBC     <OREG> <BITLIT> <ALIT>
|           CALL    <ALIT>
|           RET
|           WURM    <OREG> <OREG> <MSKLIT>
```

5.3 MCS24-3 Architecture

This MCS target architecture is selected by using the value

```
<ARCH> = mcs24-3
```

for the architecture command line switch `-arch`. ASM-MCS will assume an MCS architecture specification according to [GTMSPEC3].

5.3.1 Types

ASM-MCS provides register argument types defining a set of available symbols that can be used as register arguments. The following types are available in this architecture

```
<AREG>    ::= R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 | ZERO

<OREG>    ::= R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 | STA | ACB
            | CTRG | STRG | TBU_TS0 | TBU_TS1 | TBU_TS2 | MHB

<GREG>    ::= R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7

<XOREG>   ::= R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 | STA | ACB
            | CTRG | STRG | TBU_TS0| TBU_TS1 | TBU_TS2 | MHB
            | RS0 | RS1 | RS2 | RS3 | RS4 | RS5 | RS6 | RS7
            | GMI0 | GMI1 | DSTA
```

In order to specify literal arguments of instructions and memory initializations, the following literal types are available:

```
<WLIT>    ::= A 24 bit word literal value
              in the range [0; 2^24-1]

<ALIT>    ::= A 15 bit address literal value
              in the range [0; 2^15-1]

<AOLIT>   ::= A 15 bit address offset literal value
              in the range [-2^14-1; 2^14-1]

<ARDLIT>  ::= A 9 bit ARU read address literal value
              in the range [0; 2^8-1]

<AWRLIT>  ::= A 5  bit ARU write index literal value
              in the range [0; 23]
```

```
<SFTLIT>  ::= A 5 bit shift literal value in the range [0; 24]

<BITLIT>  ::= A 4 bit shift literal value in the range [0; 15]

<MSKLIT>  ::= A 16 bit mask literal value
              in the range [0; 2^16-1]

<BUSLIT>  ::= A 16 bit bus address literal value
              in the range [0; 2^16-1]

<BWSLIT>  ::= A 5 bit bit-width selection operand
              in the range [1; 24]
```

5.3.2 Instructions

In this architecture ASM-MCS implements the instructions set as mentioned in [GTMSPEC1]. ASM-MCS uses the following rules in a non case sensitive manner to match an instruction:

```
<INSTR>::=     NOP
     |         MOVL    <OREG> <WLIT>
     |         ADDL    <OREG> <WLIT>
     |         SUBL    <OREG> <WLIT>
     |         ANDL    <OREG> <WLIT>
     |         ORL     <OREG> <WLIT>
     |         XORL    <OREG> <WLIT>
     |         ATUL    <OREG> <WLIT>
     |         ATSL    <OREG> <WLIT>
     |         BTL     <OREG> <WLIT>
     |         MOV     <XOREG> <XOREG>
     |         MRD     <OREG> <ALIT>
     |         MWR     <OREG> <ALIT>
     |         MRDI    <OREG> <OREG> [<AOLIT>]
     |         MWRI    <OREG> <OREG> [<AOLIT>]
     |         POP     <OREG>
     |         PUSH    <OREG>
     |         MWRL    <OREG> <ALIT>
     |         MWRIL   <OREG> <OREG>
     |         ARD     <AREG> <AREG> <ARDLIT>
     |         AWR     <OREG> <OREG> <AWRLIT>
     |         ARDI    <AREG> <AREG>
     |         AWRI    <OREG> <OREG>
```

```
|          NARD    <AREG> <AREG> <ARDLIT>
|          NARDI   <AREG> <AREG>
|          BRD     <GREG> <BUSLIT>
|          BWR     <GREG> <BUSLIT>
|          BRDI    <GREG> <GREG>
|          BWRI    <GREG> <GREG>
|          ADD     <XOREG> <XOREG>
|          SUB     <XOREG> <XOREG>
|          NEG     <XOREG> <XOREG>
|          AND     <XOREG> <XOREG>
|          OR      <XOREG> <XOREG>
|          XOR     <XOREG> <XOREG>
|          SHR     <XOREG> <SFTLIT>
|          SHL     <XOREG> <SFTLIT>
|          ASRU    <XOREG> <XOREG>
|          ASRS    <XOREG> <XOREG>
|          ASL     <XOREG> <XOREG>
|          MINU    <XOREG> <XOREG>
|          MINS    <XOREG> <XOREG>
|          MAXU    <XOREG> <XOREG>
|          MAXS    <XOREG> <XOREG>
|          MULU    <XOREG> <XOREG> [<BWSLIT>]
|          MULS    <XOREG> <XOREG> [<BWSLIT>]
|          ATU     <XOREG> <XOREG>
|          ATS     <XOREG> <XOREG>
|          BT      <XOREG> <XOREG>
|          JMP     <ALIT>
|          JBS     <OREG> <BITLIT> <ALIT>
|          JBC     <OREG> <BITLIT> <ALIT>
|          CALL    <ALIT>
|          RET
|          WURM    <OREG> <OREG> <MSKLIT>
```