

The MYMIPS Assembly Language

1. Introduction

The program "mymipsasm" is provided as a rudimentary assembler for the MYMIPS machine. There is no guarantee that it works correctly for all cases, although every effort to see that it does so has been made. Please report any bugs found to raw@cs.duke.edu.

2. Language

Each statement in the mymipsasm source language, hereafter called "AL", consists of an optional "label:" first field, an "opcode" field, followed by zero or more operand fields. The label field must begin in column 1 of the line, and the label itself must contain only letters and digits, and must begin with a letter. The opcode field must NOT begin in column 1 (if not labeled, the statement should begin with whitespace).

The "label", if present is entered into the symbol table with a value equal to the next free location in memory. This value is always a byte address which is a multiple of 4, and is equal to the location into which the next word or instruction in the source text will be placed, after being translated. The appearance of a label in the "label:" context serves to "define" that label. A "label" may be used in the source text wherever a NUM is allowed, if that label has been defined somewhere in the source text. A label may not be defined more than once.

Each source line is translated into one or more MYMIPS memory words. Only special assembly directives translate into more than one word -- all ordinary instruction op-codes translate into one MYMIPS word. One exception to this is a line with a blank op-code, which translates into NO words.

The "#" character introduces commentary. The "#", and all characters after it on the same line, will be ignored by the assembler.

2.1. Instruction formats

The formats of instruction operand fields are controlled by the op-code field. Subfields of the operand field are separated by: whitespace, or ",", or "(" . A subfield separator can be followed by any amount of whitespace. In the table below, the notation R stands for a register name (which must begin with a "\$"), the notation NUM must be either a number (either decimal or hexadecimal, or even octal, following the C conventions), or a label defined in the program. The notation VAL stands for either R, or NUM.

OpCode Type	Forms Allowed	Example	
Arithmetic	OP R,R,VAL	add \$1,\$2,33	\$1 = \$2 + 33
		add \$1,\$2,\$2	\$1 = \$2+\$2
		add \$1,\$0,Label	Move address "Label" to register 1
Br Cond	OP R,VAL	bltz \$1,L1	goto L1 if \$1<0
Branch	OP VAL	bltz \$1,\$3	goto location in \$3 if \$1<0
		bal hextoint	Set \$15 to PC+4, and goto "hextoint"
		bal \$2	Set \$15 to PC+4, and goto location in \$2
		b loop	goto "loop"

Memory	OP R,NUM	lw \$1,ret	Load contents of memory location "ret" into \$1
	OP R,VAL(R)	lw \$1,ret(\$5)	Load contents of memory location ("ret"+\$5) into \$1
		lw \$1,\$2(\$3)	Load contents of memory location (\$2+\$3) into \$1

The table below shows, for each allowable instruction opcode, that instruction's "type":

Opcode	Type	Hex Opcode	D field
add	Arithmetic	5	
sub	Arithmetic	6	
sla	Arithmetic	7	
and	Arithmetic	8	
or	Arithmetic	9	
xor	Arithmetic	A	
lw	Memory	1	
lb	Memory	2	
sw	Memory	3	
sb	Memory	4	
syscall	Branch	0	
bal	Branch	F	
b	Branch	E	0
nop	Branch	E	7
bltz	Br Cond	E	1
beqz	Br Cond	E	2
blez	Br Cond	E	3
bgtz	Br Cond	E	4
bnez	Br Cond	E	5
bgez	Br Cond	E	6

2.2. Register names

Registers named \$0 through \$15 are defined, and so are the symbolic names given in the table below. Note that each register has one numeric and one symbolic name.

Numeric	Symbolic	Usage
\$0	\$zero	Holds value 0
\$1	\$v0	Return value, and caller-saved temporary
\$2	\$a0	Argument value, and caller-saved temporary
\$3	\$a1	Argument value, and caller-saved temporary
\$4	\$t0	Caller-saved temporary
\$5	\$t1	Caller-saved temporary
\$6	\$t2	Caller-saved temporary
\$7	\$s0	Callee-saved temporary
\$8	\$s1	Callee-saved temporary
\$9	\$s2	Callee-saved temporary
\$10	\$s3	Callee-saved temporary
\$11	\$s4	Callee-saved temporary
\$12	\$s5	Callee-saved temporary
\$13	\$s6	Callee-saved temporary
\$14	\$sp	Stack pointer
\$15	\$ra	Return address

2.3. Assembler directives

These commands are used to place words of data into memory, or to end the source text.

- .asciiz The operand field consists of a single string of non-whitespace characters. It is loaded into memory beginning on a word boundary. Within the string, any "special" character can be represented by preceding that character with "\". The C notations for NEWLINE, TAB, and NULL are available. The characters SPACE, ",", "(", and ":" also terminate the string, and must be preceded with a "\" if they are to be included inside the string.
- .word The operand field is a single NUM, whose value is loaded into the next free word of memory.
- .space The operand field is a number, which gives the number of bytes to skip. In fact, this translates into an even number of words of zeros, which fill the requested number of bytes, plus enough to set the next load location to an even word boundary.
- .end The .end directive marks the end of the source text. Its operand field is a VAL, which becomes the starting address of the program. This must be a byte address which is a multiple of 4.

3. An example

Below is the "assembly listing" of a valid source program. The columns labeled "SOURCE TEXT", extending to the end of each line, represent the input to the assembler; The column labeled LOC indicates the memory location into which each translated word will be loaded, and the column labeled TOLOADER is the actual output from the assembler, to be read by the mymips loader. This listing was produced by hand -- the assembler produces only the column labeled TOLOADER.

TOLOADER LOC SOURCE TEXT

```
# Program to read numbers until a zero,
# printing each in hexadecimal
00000000 00000 ret: .word 0 # space for return address
00000000 00004 s: .space8
00000000 00008
0A000000 0000C .asciiz
30313233 00010 tr: .asciiz 0123456789ABCDEF
34353637 00014
38394142 00018
43444546 0001C
00000000 00020
3F080000 00024 main: sw $ra,ret # save return address

00080005 00028 syscall 5 # read number
E2180044 0002C beqz $v0,exit # if number is 0, exit
52100000 00030 add $a0,$v0,$0 # copy number to arg register
F0080050 00034 bal itohex # convert to hex in s[]
52080004 00038 add $a0,$0,s # prepare to print s[]
00080004 0003C syscall 4 # print the string
E0080028 00040 b ml1 # repeat

51000000 00044 exit: add $v0,$0,$0 # zero exit status
1F080000 00048 lw $ra,ret # restore return address
0008000A 0004C syscall 10 # return from main

56080004 00050 itohex: add $t2,$0,s # $t2 = &s[0] (used as end test)
54680007 00054 add $t0,$t2,7 # $t0 = &s[7] (k)
8528000F 00058 loop: and $t1,$a0,0xF # $t1 = I & 0xF
25580010 0005C lb $t1,tr($t1) # $t1 = tr[I&0xF]
45480000 00060 sb $t1,0($t0) # s[k] = $t1
722FFFFC 00064 sla $a0,$a0,-4 # I = I >> 4
544FFFFF 00068 add $t0,$t0,-1 # k--
63400006 0006C sub $a1,$t0,$t2 # -->loop if k>=0
E6380058 00070 bgez $a1,loop # -->loop if k>=0
E000000F 00074 b $ra # return
-1 00024 .end main
```