

MIPS Assembler 简介

1、所支持的指令：

R					
add	rd, rs, rt	10 0000	0x20		
addu	rd, rs, rt	10 0001	0x21		
and	rd, rs, rt	10 0100	0x24		
break		00 1101	0x0D		
div	rs, rt	01 1010	0x1A		
divu	rs, rt	01 1011	0x1B		
jalr	rd, rs	00 1001	0x09		
jr	rs	00 1000	0x08		
mfhi	rd	01 0000	0x10		
mflo	rd	01 0010	0x12		
mthi	rs	01 0001	0x11		
mtlo	rs	01 0011	0x13		
mult	rs, rt	01 1000	0x18		
multu	rs, rt	01 1001	0x19		
nor	rd, rs, rt	10 0111	0x27		
or	rd, rs, rt	10 0101	0x25		
sll	rd, rt, sa	00 0000	0x00		
sllv	rd, rt, rs	00 0100	0x04		
slt	rd, rs, rt	10 1010	0x2A		
sltu	rd, rs, rt	10 1011	0x2B		
sra	rd, rt, sa	00 0011	0x03		
srav	rd, rt, rs	00 0111	0x07		
srl	rd, rt, sa	00 0010	0x02		
srlv	rd, rt, rs	00 0110	0x06		
sub	rd, rs, rt	10 0010	0x22		
subu	rd, rs, rt	10 0011	0x23		
syscall		00 1100	0x0C		
xor	rd, rs, rt	10 0110	0x26		
I					
addi	rt, rs, immediate	00 1000	0x08		
addiu	rt, rs, immediate	00 1001	0x09		
andi	rt, rs, immediate	00 1100	0x0C		
beq	rs, rt, label	00 0100	0x04		
bgez	rs, label	00 0001	0x01	rt = 00001	
bgtz	rs, label	00 0111	0x07	rt = 00000	
blez	rs, label	00 0110	0x06	rt = 00000	
bltz	rs, label	00 0001	0x01	rt = 00000	

bne	rs, rt, label	00 0101 0x05
lb	rt, immediate(rs)	10 0000 0x20
lbu	rt, immediate(rs)	10 0100 0x24
lh	rt, immediate(rs)	10 0001 0x21
lhu	rt, immediate(rs)	10 0101 0x25
lui	rt, immediate	00 1111 0x0F
lw	rt, immediate(rs)	10 0011 0x23
lwc1	rt, immediate(rs)	11 0001 0x31
ori	rt, rs, immediate	00 1101 0x0D
sb	rt, immediate(rs)	10 1000 0x28
slti	rt, rs, immediate	00 1010 0x0A
sltiu	rt, rs, immediate	00 1011 0x0B
sh	rt, immediate(rs)	10 1001 0x29
sw	rt, immediate(rs)	10 1011 0x2B
swc1	rt, immediate(rs)	11 1001 0x39
xori	rt, rs, immediate	00 1110 0x0E
J		
j	label 000010	coded address of label 0x02
jal	label 000011	coded address of label 0x03

其中, syscall 的更进一步实现尚不支持, 但是支持将这一指令翻译为机器码。

除此之外, 本汇编器支持标号, 有较完备的错误提示系统。

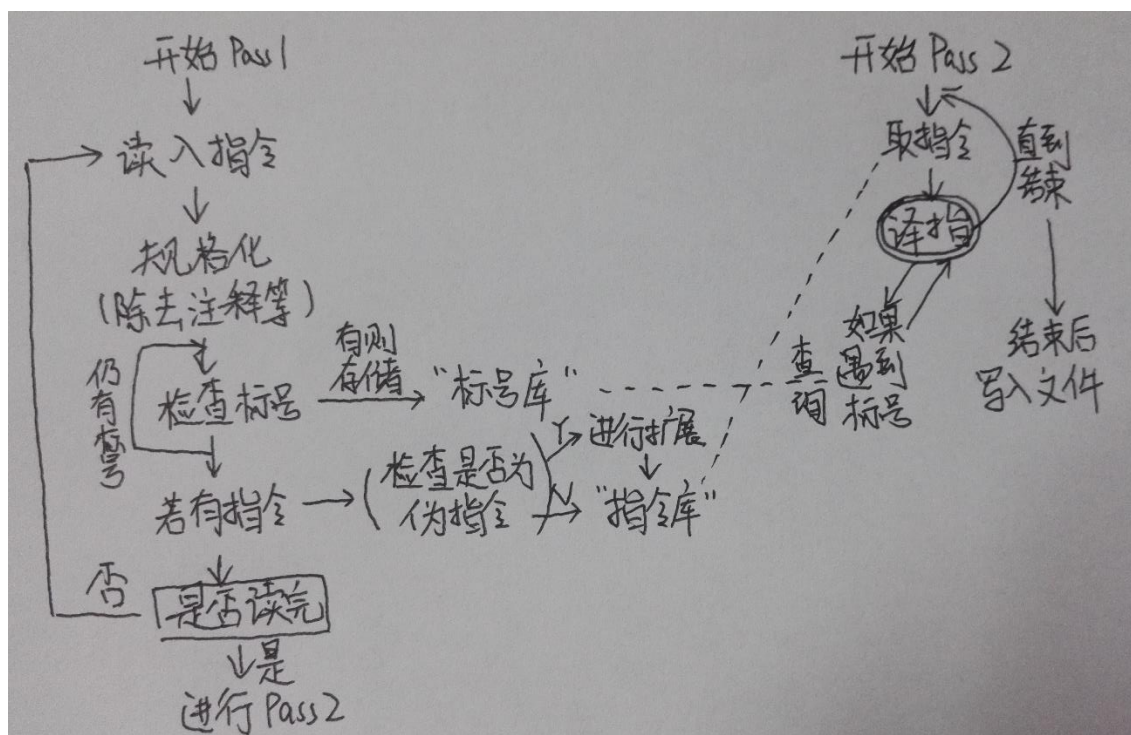
目前除部分指令外尚不支持的功能有: 表达式以及伪指令, 不过在本程序的源代码中为伪指令的实现以及表达式的支持预留了接口, 且可以方便的增加指令, 扩展程序支持的指令集。

要使本程序支持伪指令, 只需写 translate.c 文件中的 `AsmCode getRealInstrLine(int32 psedoSerNum, uint32 lineNum)` 函数并将 basic.h 文件中的宏定义 `PSEDO_MODE` 定义为 1 即可。要使本程序支持表达式, 只需对 translate.c 文件中的 `AsmCode clearInBracSpace(AsmCode line)` 函数进行加强, 使得在 '\$' 子串前留出空格, 并重写 assembler.c 文件中的 `int32 getImme(char* labelOrImme, AsmRecorder* asmRecorderPtr)` 函数, 随后对 `void genMachineCode(AsmRecorder* asmRecorderPtr)` 函数的静态变量 `delimiters` 进

行修改即可；要增加指令，只需在 translate.h 文件的宏定义中增加相应的项并修改对应的宏定义 XX_NUM（实际上不修改并不会引发错误），随后在 assmebler.c 文件中的 void genMachineCode(AsmRecorder* asmRecorderPtr) 函数中增添相应的项即可，若增加的是伪指令，则应去修改 translate.c 文件中的 AsmCode getRealInstrLine(int32 psedoSerNum, uint32 lineNum) 函数。

2、程序框图：

因为需要处理标号，本程序采取两遍扫描，第一遍记录标号并读取指令，第二遍翻译指令为机器码，当指令中存在标号的时候则去查询标号并进行相应的计算，具体流程如下（以正常流程为例）：



其中指令到机器码的翻译是编译的关键一步，以下为伪代码叙述（以目前所支持的指令为例，依然不考虑出错情况）：

```
void genMachineCode(AsmRecorder* asmRecorderPtr)
{
```

```

#define GET_REGISTER(rx) \
    registerName = strtok(NULL, delimiters); \
    rx = getRegisterSerNum(registerName);
#define CHECKEND
#define GET_LABELORINNE \
    labelOrImme = strtok(NULL, delimiters); \
    immeOrOffset = getImmeOrOffset(labelOrImme, asmRecorderPtr);
#define GET_IMME \
    labelOrImme = strtok(NULL, delimiters); \
    immeOrOffset = getImme(labelOrImme, asmRecorderPtr);

static char* delimiters = " \\t,()";
char* instr = asmRecorderPtr->instruction;
Mnemonic mnemonic = strtok(instr, delimiters);
int32 mnemonicSerNum = getMnemonicSerNum(mnemonic);
char *registerName;
if (mnemonicSerNum < R_NUM)
{
    uint32 rd, rs, rt;
    int32 schamt;
    rd = rs = rt = schamt = 0x0;
    asmRecorderPtr->code = getFuncOrOpcode(mnemonicSerNum);
    switch (asmRecorderPtr->code)
    {
        case 0x20:
        case 0x21:
        case 0x24:
        case 0x27:
        case 0x25:
        case 0x2A:
        case 0x2B:
        case 0x22:
        case 0x23:
        case 0x26: // dst
            GET_REGISTER(rd)
            GET_REGISTER(rs)
            GET_REGISTER(rt)
            CHECKEND
            break;
        case 0x00:
        case 0x03:
        case 0x02: // dta
            GET_REGISTER(rd)
            GET_REGISTER(rt)

```

```
    char* schamtStr = strtok(NULL, delimiters);
    schamt = getImme(schamtStr, asmRecorderPtr);
    schamt &= 0x1F;
    CHECKEND
    break;
case 0x04:
case 0x07:
case 0x06: // dts
    GET_REGISTER(rd)
    GET_REGISTER(rt)
    GET_REGISTER(rs)
    CHECKEND
    break;
case 0x09: // ds
    GET_REGISTER(rd)
    GET_REGISTER(rs)
    CHECKEND
    break;
case 0x1A:
case 0x1B:
case 0x18:
case 0x19: // st
    GET_REGISTER(rs)
    GET_REGISTER(rt)
    CHECKEND
    break;
case 0x08:
case 0x11:
case 0x13: // s
    GET_REGISTER(rs)
    CHECKEND
    break;
case 0x10:
case 0x12: // d
    GET_REGISTER(rd)
    CHECKEND
    break;
case 0x0D:
case 0x0C: // 0
    CHECKEND
    break;
}
asmRecorderPtr->code |= (rs << 21);
asmRecorderPtr->code |= (rt << 16);
```

```
asmRecorderPtr->code |= (rd << 11);
asmRecorderPtr->code |= (schamt << 6);
return;
}
else
{
    uint32 rt, rs;
    char* labelOrImme;
    int32 immeOrOffset;
    rt = rs = immeOrOffset = 0x0;
    asmRecorderPtr->code = getFuncOrOpcode(mnemonicSerNum) <<
26;

    switch (asmRecorderPtr->code)
    {
    case 0x02 << 26:
    case 0x03 << 26: // J
        GET_LABELORINNE
        CHECKEND
        immeOrOffset &= 0x3FFFFFF;
        asmRecorderPtr->code |= immeOrOffset;
        return;
    case 0x08 << 26:
    case 0x09 << 26:
    case 0x0C << 26:
    case 0x0D << 26:
    case 0x0A << 26:
    case 0x0B << 26:
    case 0x0E << 26: // rt, rs, immediate
        GET_REGISTER(rt)
        GET_REGISTER(rs)
        GET_IMME
        CHECKEND
        immeOrOffset &= 0xFFFF;
        break;
    case 0x04 << 26:
    case 0x05 << 26: // rs, rt, label
        GET_REGISTER(rs)
        GET_REGISTER(rt)
        GET_LABELORINNE
        CHECKEND
        immeOrOffset &= 0xFFFF;
        break;
    case 0x01 << 26:
        if (strcmp(mnemonic, "bgez") == 0)
```

```
    {
        rt = 0x1;
    }
    case 0x07 << 26:
    case 0x06 << 26: // rs, label
        GET_REGISTER(rs)
        GET_LABELORINNE
        CHECKEND
        immeOrOffset &= 0xFFFF;
        break;
    case 0x20 << 26:
    case 0x24 << 26:
    case 0x21 << 26:
    case 0x25 << 26:
    case 0x23 << 26:
    case 0x31 << 26:
    case 0x28 << 26:
    case 0x29 << 26:
    case 0x2B << 26:
    case 0x39 << 26: // rt, immediate(rs)
        GET_REGISTER(rt)
        GET_IMME
        GET_REGISTER(rs)
        CHECKEND
        immeOrOffset &= 0xFFFF;
        break;
    case 0x0F << 26: // rt, immediate
        GET_REGISTER(rt)
        GET_IMME
        CHECKEND
        immeOrOffset &= 0xFFFF;
        break;
    }
    asmRecorderPtr->code |= immeOrOffset;
    asmRecorderPtr->code |= (rt << 16);
    asmRecorderPtr->code |= (rs << 21);
    return;
}

#undef GET_REGISTER
#undef CHECKEND
#undef GET_SYMBOLORIMME
#undef GET_IMME
}
```

3、使用方法：

在命令行中不带参数直接执行程序或执行程序参数有误时，即可获得使用方法，如下：

```
>asmer.exe  
Usage:  
asmer.exe <filename1> <filename2>  
filename1: the name/address of the input file.  
filename2: the name/address of the output file.  
e.g. asmer.exe input.asm output.bin  
By: Haotian_Ren 3150104714
```

本程序中的错误信息目前共有以下 16 种：

- | | |
|----|---|
| 1 | Mallocating failed. |
| 2 | Unable to open file %s |
| 3 | Unable to create file %s |
| 4 | Line %d: Too many symbols. |
| 5 | Line %d: Symbol '%s' has been defined in line %d. |
| 6 | Line %d: Wrong mnemonic. |
| 7 | Line %d: The line is too long. |
| 8 | Line %d: Brackets not matched. |
| 9 | Line %d: Wrong register name '%s'. |
| 10 | Line %d: Wrong instruction format. |
| 11 | Line %d: Unsupported mnemonic. |
| 12 | Line %d: Symbol '%s' is not defined. |
| 13 | Line %d: '%s' is not an integer. |
| 14 | Line %d: Wrong symbol format. |
| 15 | Line %d: Symbol is too long. |
| 16 | Too many lines. |

4、实例分析

在此列举一例实例，以下仅进行运行，不讨论汇编代码具体含义，汇编代码如下（test.asm）：

1	# Test function over here:
2	addiu \$a0, \$0, -25 # Load base addr
3	addiu \$a1, \$0, 10 # Load length
4	jal MYFUNC # Call function
5	li \$v0, 0xABCDE


```

6
7 myFunc: addiu $t0, $0, 0      # Init counter
8 startLoop: beq $t0, $a1, endLoop    # Check if end condition reached
9     addu $t1, $a0, $t0      # Calculate offset address
10    lb $t2, 0($t1)          # Load value
11
12    lbu $t3, -3($s2)
13    addiu $t2, $t2, 1        # Increment by one
14    or $a0, $a1, $a3
15    li $t0, 3
16    slt $a2, $t1, $t0
17    sltu $a2, $t1, $t0
18    sll $t3, $t2, 31
19
20 random: ori $t3, $t2, 291
21    lui $t3, 532
22    sb $t2, 0($t1)
23    sw $t2, -32768($t1)
24    lw $t3, 32767($t1)
25    blt $t3, $t2, myFunc
26
27    addiu $t1, $t1, 1        # Increment counter
28    j startLoop
29 endLoop: jr $ra
30    bne $t3, $a0, myFunc
31

```

执行结果如下图所示：

```

>asmer.exe test.asm testout.bin
Error: Line 5: Unsupported mnemonic.

```

我们可以注释掉第 5、15、25 行不支持的三个指令，则程序的运行结果如

下图所示：

```

>asmer.exe test.asm testout.bin
Done!

```

程序输出“Done!”代表已经汇编成功，我们可以打开生成的二进制文件查看

其中的内容，如下所示：

```

e7ff 0424 0a00 0524 0100 000c 0000 0824
1000 0511 2148 8800 0000 2a81 fdff 4b92
0100 4a25 2520 a700 2a30 2801 2b30 2801
c05f 0a00 2301 4b35 1402 0b3c 0000 2aa1

```

0080 2aad ff7f 2b8d 0100 2925 f1ff ff0b 0800 e003 eeff 6415
--

对照后我们亦可得出进行汇编过程所使用机器，即 asmer.exe 所运行机器使用小端存储的方式存储数据。

我们最终所使用的测试 MIPS 汇编代码如下：

```
1  # Test function over here:
2      addiu $a0, $0, -25      # Load base addr
3      addiu $a1, $0, 10      # Load length
4      jal MYFUNC             # Call function
5      #li $v0, 0xABCDE
6
7 myFunc: addiu $t0, $0, 0      # Init counter
8 startLoop: beq $t0, $a1, endLoop      # Check if end condition reached
9      addu $t1, $a0, $t0      # Calculate offset address
10     lb $t2, 0($t1)          # Load value
11
12     lbu $t3, -3($s2)
13     addiu $t2, $t2, 1        # Increment by one
14     or $a0, $a1, $a3
15     #li $t0, 3
16     slt $a2, $t1, $t0
17     sltu $a2, $t1, $t0
18     sll $t3, $t2, 31
19
20 random: ori $t3, $t2, 291
21     lui $t3, 532
22     sb $t2, 0($t1)
23     sw $t2, -32768($t1)
24     lw $t3, 32767($t1)
25     #blt $t3, $t2, myFunc
26
27     addiu $t1, $t1, 1        # Increment counter
28     j startLoop
29 endLoop: jr $ra
30     bne $t3, $a0, myFunc
31
```

MIPS Disassembler 简介

1、所支持的反汇编指令：

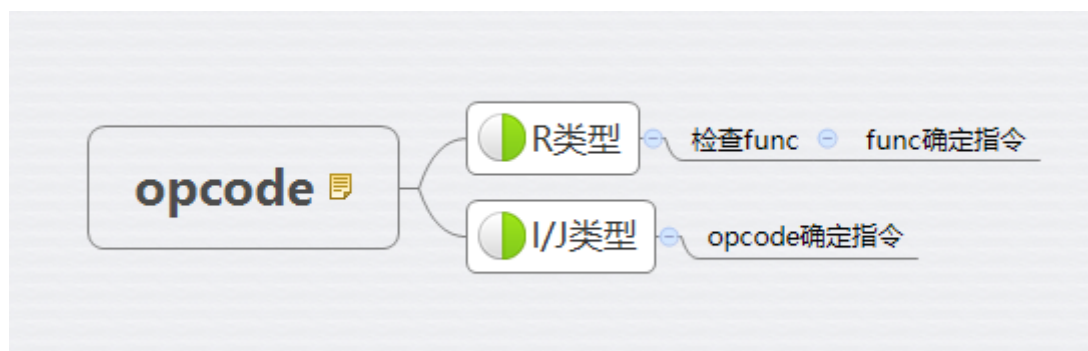
R					
add	rd, rs, rt	10 0000	0x20		
addu	rd, rs, rt	10 0001	0x21		
and	rd, rs, rt	10 0100	0x24		
break		00 1101	0x0D		
div	rs, rt	01 1010	0x1A		
divu	rs, rt	01 1011	0x1B		
jalr	rd, rs	00 1001	0x09		
jr	rs	00 1000	0x08		
mfhi	rd	01 0000	0x10		
mflo	rd	01 0010	0x12		
mthi	rs	01 0001	0x11		
mtlo	rs	01 0011	0x13		
mult	rs, rt	01 1000	0x18		
multu	rs, rt	01 1001	0x19		
nor	rd, rs, rt	10 0111	0x27		
or	rd, rs, rt	10 0101	0x25		
sll	rd, rt, sa	00 0000	0x00		
sllv	rd, rt, rs	00 0100	0x04		
slt	rd, rs, rt	10 1010	0x2A		
sltu	rd, rs, rt	10 1011	0x2B		
sra	rd, rt, sa	00 0011	0x03		
srav	rd, rt, rs	00 0111	0x07		
srl	rd, rt, sa	00 0010	0x02		
srlv	rd, rt, rs	00 0110	0x06		
sub	rd, rs, rt	10 0010	0x22		
subu	rd, rs, rt	10 0011	0x23		
syscall		00 1100	0x0C		
xor	rd, rs, rt	10 0110	0x26		
I					
addi	rt, rs, immediate	00 1000	0x08		
addiu	rt, rs, immediate	00 1001	0x09		
andi	rt, rs, immediate	00 1100	0x0C		
beq	rs, rt, label	00 0100	0x04		
bgez	rs, label	00 0001	0x01	rt = 00001	
bgtz	rs, label	00 0111	0x07	rt = 00000	
blez	rs, label	00 0110	0x06	rt = 00000	
bltz	rs, label	00 0001	0x01	rt = 00000	

bne	rs, rt, label	00 0101 0x05
lb	rt, immediate(rs)	10 0000 0x20
lbu	rt, immediate(rs)	10 0100 0x24
lh	rt, immediate(rs)	10 0001 0x21
lhu	rt, immediate(rs)	10 0101 0x25
lui	rt, immediate	00 1111 0x0F
lw	rt, immediate(rs)	10 0011 0x23
lwc1	rt, immediate(rs)	11 0001 0x31
ori	rt, rs, immediate	00 1101 0x0D
sb	rt, immediate(rs)	10 1000 0x28
slti	rt, rs, immediate	00 1010 0x0A
sltiu	rt, rs, immediate	00 1011 0x0B
sh	rt, immediate(rs)	10 1001 0x29
sw	rt, immediate(rs)	10 1011 0x2B
swc1	rt, immediate(rs)	11 1001 0x39
xori	rt, rs, immediate	00 1110 0x0E
J		
j	label	000010 coded address of label 0x02
jal	label	000011 coded address of label 0x03

本反汇编器所支持的指令在代码开始处的宏进行统一管理，由机器码到汇编代码的翻译在 `void disassembleNode(InstrNode* instrNode)` 函数中进行，因此若要扩展反汇编器所支持的指令，只需对代码开头的宏进行修改并在 `void disassembleNode(InstrNode* instrNode)` 函数中增加相应的处理即可。

2、算法逻辑：

反汇编的算法较为简单，大致可以描述如下：



3、使用方法：

在命令行中不带参数直接执行程序或执行程序参数有误时，即可获得使用方法，如下：

```
>disassembler.exe
Disassembler Usage:
disassembler.exe <filename1> <filename2>
filename1: the name/address of the input file.
filename2: the name/address of the output file.
e.g. disassembler.exe input.bin output.asm
By: Haotian Ren (3150104714@zju.edu.cn)
```

4、实例分析

首先使用汇编器程序将汇编代码转化为机器码，随后再使用反汇编器对机器码进行反汇编。所使用的测试汇编代码如下：

```
1  # Test function over here:
2      addiu $a0, $0, -25      # Load base addr
3      addiu $a1, $0, 10      # Load length
4      jal MYFUNC             # Call function
5      li $v0, 0xABCDE
6
7  myFunc: addiu $t0, $0, 0      # Init counter
8  startLoop: beq $t0, $a1, endLoop      # Check if end condition reached
9      addu $t1, $a0, $t0      # Calculate offset address
10     lb $t2, 0($t1)          # Load value
11
12     lbu $t3, -3($s2)
13     addiu $t2, $t2, 1        # Increment by one
14     or $a0, $a1, $a3
15     li $t0, 3
16     slt $a2, $t1, $t0
17     sltu $a2, $t1, $t0
18     sll $t3, $t2, 31
19
20  random: ori $t3, $t2, 291
21     lui $t3, 532
22     sb $t2, 0($t1)
23     sw $t2, -32768($t1)
24     lw $t3, 32767($t1)
25     blt $t3, $t2, myFunc
26
27     addiu $t1, $t1, 1        # Increment counter
28     j startLoop
29  endLoop: jr $ra
30     bne $t3, $a0, myFunc
31
```

对由该汇编代码生成的机器码进行反汇编的结果如下：

```
>disassembler.exe test.bin out.asm
Done!
```

程序输出“Done!”表示反汇编成功。

在 out.asm 文件中可以查看生成的汇编代码，如下：

```
>type out.asm
addiu $a0, $zero, -25    # MachineCode: 0x2404FFE7
addiu $a1, $zero, 10     # MachineCode: 0x2405000A
jal 1    # MachineCode: 0x0C000001
addiu $t0, $zero, 0      # MachineCode: 0x24080000
beq $t0, $a1, 16         # MachineCode: 0x11050010
addu $t1, $a0, $t0       # MachineCode: 0x00884821
lb $t2, 0($t1) # MachineCode: 0x812A0000
lbu $t3, -3($s2)         # MachineCode: 0x924BFFFD
addiu $t2, $t2, 1        # MachineCode: 0x254A0001
or $a0, $a1, $a3         # MachineCode: 0x00A72025
slt $a2, $t1, $t0        # MachineCode: 0x0128302A
sltu $a2, $t1, $t0       # MachineCode: 0x0128302B
sll $t3, $t2, 15         # MachineCode: 0x000A5FC0
ori $t3, $t2, 291        # MachineCode: 0x354B0123
lui $t3, 532             # MachineCode: 0x3C0B0214
sb $t2, 0($t1) # MachineCode: 0xA12A0000
sw $t2, -32768($t1)      # MachineCode: 0xAD2A8000
lw $t3, 32767($t1)       # MachineCode: 0x8D2B7FFF
addiu $t1, $t1, 1        # MachineCode: 0x25290001
j -15 # MachineCode: 0x0BFFFFFF1
jr $ra # MachineCode: 0x03E00008
bne $t3, $a0, -18        # MachineCode: 0x1564FFEE
```

说明反汇编结果正确。