编译器设计

实验指导书

任课老师：潘志铭

作为一种开源指令集，RISC-V 的特点是模块化设计、扩展性强，当标准的 RISC-V 扩展指令无法满足客户需求时，就需要在 RISC-V 规范允许范围内增加自定义指令集。当我们在硬件上实现了这些自定义指令后，为了使用这些自定义指令集，我们还需要修改工具链，本文将举例说明如何为 GNU 工具链增加 RISC-V 自定义指令的支持。在修改 GNU 工具链之前，我们需要预先下载 riscv-gnu-toolchain 和 riscv-tools，具体步骤为：

```
$ git clone https://github.com/riscv/riscv-gnu-toolchain.git --recursive
$ git clone https://github.com/riscv/riscv-tools.git --recursive
```

1. 描述需要增加的自定义指令

假定我们需要增加一条乘加指令 **mac**，具体描述如下：

汇编指令	mac rd, rs1, rs2
软件模型	rd = rd + rs1 * rs2
例子	mac a2, a0, a1

2. 确定指令码

依据 RISC-V SPEC 的描述，**mac** 指令属于 R-Type 指令，所以指令码大致格式为：

```
 31        25 24       20 19       15 14   12 11       7 6         0
-------------------------------------------------------------------------
|    funct7  |    rs2    |    rs1    | funct3 |    rd    |    OPCODE  |
-------------------------------------------------------------------------
       7          5           5         3          5           7
```

RISC-V SPEC 规定了四种自定义指令的 OPCODE，分别是 7'b000_1011，7'b010_1011, 7'b101_1011, 7'b111_1011, 这里我们选用 CUSTOM0 编码 7'b000_1011，为了进一步规范自定义指令码格式，我们可以约定 funct3 的编码格式：3'b010 表示只使用了 RS1 寄存器，3'b011 表示只使用了 RS1 和 RS2 寄存器，3'b100 表示只使用了 RD 寄存器，3'b110 表示只使用 RD 和 RS1 寄存器，而 3'b111 则表示 RD、RS1 和 RS2 寄存器都用到了，所以这里的 funct3 应该是 3'b111.由于 funct7 没有特别的约束，我们这里选定为 7'b101_0111.

inst[4:2] inst[6:5]	000	001	010	011	100	101	110	111 ($>32b$)
00	LOAD	LOAD-FP	*custom-0*	MISC-MEM	OP-IMM	AUIPC	OP-IMM-32	$48b$
01	STORE	STORE-FP	*custom-1*	AMO	OP	LUI	OP-32	$64b$
10	MADD	MSUB	NMSUB	NMADD	OP-FP	*reserved*	*custom-2/rv128*	$48b$
11	BRANCH	JALR	*reserved*	JAL	SYSTEM	*reserved*	*custom-3/rv128*	$\geq 80b$

funct3	rd	rs1	rs2	Symbol
3'b000	0	0	0	CUSTOMX
3'b001	0	0	1	保留
3'b010	0	1	0	CUSTOMX_RS1
3'b011	0	1	1	CUSTOMX_RS1_RS2
3'b100	1	0	0	CUSTOMX_RD
3'b101	1	0	1	保留
3'b110	1	1	0	CUSTOMX_RD_RS1
3'b111	1	1	1	CUSTOMX_RD_RS1_RS2

最终选定的编码如下：

```
 31        25 24       20 19       15 14   12 11       7 6         0
 ------------------------------------------------------------------------
|    funct7  |    rs2    |    rs1    | funct3 |    rd    |    OPCODE  |
 ------------------------------------------------------------------------
   7'b1010111      RS2         RS1       3'b111      RD       7'b0001011
        7          5           5          3          5           7
```

3. 为 binutils 增加自定义指令

3.1 编写 mac 指令对应的 MATCH 和 MASK 宏定义：

汇编器需要定义一些宏来确定指令编码，其中有两个宏是一定要定义的，分别是 MATCH 和 MASK，以 **LUI** 和 **AUIPC** 指令为例，

imm[31:12]	rd	0110111	LUI
imm[31:12]	rd	0010111	AUIPC

它们对应的 MACTH 和 MASK 宏为：

```
#define MATCH_LUI 0x37
#define MASK_LUI  0x7f
#define MATCH_AUIPC 0x17
#define MASK_AUIPC  0x7f
```

对于**LUI**和**AUIPC**指令，由于指令码的inst[31:7]是立即数，都是随机的，所以无法区分**LUI**和**AUIPC**指令，唯一能够区分两者的是inst[6:0]，所以MATCH_LUI和MATCH_AUIPC特定指明了两者的inst[6:0]编码，而MASK_LUI和MASK_AUIPC则是用于告知汇编器需要关注inst[6:0]编码，那么在生成指令码时，汇编器会这样使用这些宏定义：((insn ^ MATCH) & MASK) == 0. 如果不想手写这些宏定义的话，也可以借助github上的riscv-opcodes（https://github.com/riscv/riscv-opcodes.git）生成对应的MATCH和MASK宏定义。

3.2 修改 riscv-binutils 中的 riscv-opc.c 和 riscv-opc.h 文件：

将 MATCH 和 MASK 宏定义放在 riscv-opc.h 文件中，同时还需要在 riscv-opc.c 文件

中描述 **mac** 指令。方便起见，我们暂且把 **mac** 指令归为 M 类指令。

```
diff --git a/include/opcode/riscv-opc.h b/include/opcode/riscv-opc.h
index 7bdc7e4..c68d1c7 100644
--- a/include/opcode/riscv-opc.h
+++ b/include/opcode/riscv-opc.h
@@ -644,6 +644,9 @@
 #define MATCH_VGHASH_V 0x4a0fa057
 #define MASK_VGHASH_V  0xfe0ff07f

+#define MATCH_MAC 0xae00700b
+#define MASK_MAC  0xfe00707f
 /* Temporary Load/store encoding info
 MOP load
 00 unit-stride VLE<EEW>, VLE<EEW>FF, VL<nf>RE<EEW> (nf = 1, 2, 4, 8)
diff --git a/opcodes/riscv-opc.c b/opcodes/riscv-opc.c
index 43714eb..26a3db2 100644
--- a/opcodes/riscv-opc.c
+++ b/opcodes/riscv-opc.c
@@ -897,6 +897,7 @@ const struct riscv_opcode riscv_opcodes[] =
 {"divuw",    64, INSN_CLASS_M, "d,s,t",  MATCH_DIVUW, MASK_DIVUW,
match_opcode, 0 },
 {"remw",     64, INSN_CLASS_M, "d,s,t",  MATCH_REMW, MASK_REMW,
match_opcode, 0 },
 {"remuw",    64, INSN_CLASS_M, "d,s,t",  MATCH_REMUW, MASK_REMUW,
match_opcode, 0 },
+{"mac",       0, INSN_CLASS_M, "d,s,t",  MATCH_MAC, MASK_MAC, match_opcode,
0 },

 /* Half-precision floating-point instruction subset */
 {"flh",        0, INSN_CLASS_F_AND_ZFH,   "D,o(s)",  MATCH_FLH, MASK_FLH,
match_opcode, INSN_DREF|INSN_2_BYTE },
```

4. 为 gcc 增加自定义指令

为了让 GCC 识别 **mac** 指令，我们需要在 GCC 后端添加指令模板，由于 **mac** 指令可以描述成乘、加两种指令的混合体，所以我们可以直接使用 GCC 中自带的 SPN 来描述指令模板（对于 AES、SHA 这种加密指令，是无法直接使用 GCC 中自带的 SPN 来描述的，所以只能使用 intrinsic，然后通过 intrinsic 与 GCC 后端的对应指令模板做匹配），经过修改后的 riscv.md 文件如下：

```
diff --git a/gcc/config/riscv/riscv.md b/gcc/config/riscv/riscv.md
index 8cfac79..c1b72ee 100644
--- a/gcc/config/riscv/riscv.md
+++ b/gcc/config/riscv/riscv.md
@@ -750,6 +750,25 @@
   [(set_attr "type" "imul")
    (set_attr "mode" "SI")])

+(define_insn "macsi3"
+  [(set (match_operand:SI          0 "register_operand" "=r")
+        (plus: SI (mult:SI (match_operand:SI 2 "register_operand" " r")
+                           (match_operand:SI 3 "register_operand" " r"))
+                  (match_operand: SI 1 "register_operand" "0")))]
+  "TARGET_MUL"
+  "mac\t%0,%2,%3"
+  [(set_attr "type" "imul")
+   (set_attr "mode" "SI")])
+
+(define_insn "macdi3"
+  [(set (match_operand:DI          0 "register_operand" "=r")
+        (plus: DI (mult:DI (match_operand:DI 2 "register_operand" " r")
+                           (match_operand:DI 3 "register_operand" " r"))
+                  (match_operand: DI 1 "register_operand" "0")))]
+  "TARGET_MUL && TARGET_64BIT"
+  "mac\t%0,%2,%3"
+  [(set_attr "type" "imul")
+   (set_attr "mode" "DI")])
 ;;
 ;;  ....................
 ;;
```

5. 在 spike 上增加自定义指令

spike 是一款 RISC-V 指令模拟器，我们可以在 spike 上增加自定义指令，运行包含自定义指令 mac 的应用程序，从而验证修改后的 GNU 工具链是否能够正常工作。

5.1 更新 riscv/encoding.h 文件：

```
diff --git a/riscv/encoding.h b/riscv/encoding.h
index 9cbb271..42391d2 100644
--- a/riscv/encoding.h
+++ b/riscv/encoding.h
@@ -1844,6 +1844,10 @@
 #define MASK_VL4R_V  0xfff0707f
 #define MATCH_VL8R_V 0x1e807007
 #define MASK_VL8R_V  0xfff0707f
+#define MATCH_MAC 0xae00700b
+#define MASK_MAC  0xfe00707f
 #define CSR_FFLAGS 0x1
 #define CSR_FRM 0x2
 #define CSR_FCSR 0x3
@@ -2921,6 +2925,7 @@ DECLARE_INSN(vl1r_v, MATCH_VL1R_V, MASK_VL1R_V)
 DECLARE_INSN(vl2r_v, MATCH_VL2R_V, MASK_VL2R_V)
 DECLARE_INSN(vl4r_v, MATCH_VL4R_V, MASK_VL4R_V)
 DECLARE_INSN(vl8r_v, MATCH_VL8R_V, MASK_VL8R_V)
+DECLARE_INSN(mac, MATCH_MAC, MASK_MAC)
 #endif
 #ifdef DECLARE_CSR
 DECLARE_CSR(fflags, CSR_FFLAGS)
```

5.2 增加指令描述：

在 riscv/insns 目录下新建文件 mac.h，内容如下：

```
require_extension('M');
WRITE_RD(sext_xlen(RD + RS1 * RS2));
```

修改 riscv/riscv.mk.in 文件：

```
diff --git a/riscv/riscv.mk.in b/riscv/riscv.mk.in
index d4422fe..aa6f68b 100644
--- a/riscv/riscv.mk.in
+++ b/riscv/riscv.mk.in
@@ -195,6 +195,7 @@ riscv_insn_ext_m = \
        remu \
        remuw \
        remw \
+       mac \

 riscv_insn_ext_f = \
        fadd_s \
```

6. 构建、测试修改 GNU 工具链和 spike

在完成上面的修改后，需要重新构建 GNU 工具链和 spike，在构建过程中，请确保您具有 root 权限。

6.1 重新构建 GNU 工具链：

$ cd riscv-gnu-toolchain
$ mkdir build && cd build

```
$ ../configure --with-arch=rv64gc --with-abi=lp64d
$ make -j$(nproc)
```

6.2 重新构建 spike 和 pk:

```
$ cd riscv-isa-sim
$ mkdir build && cd build
$ ../configure
$ make -j$(nproc) && make install
$ cd riscv-pk
$ mkdir build && cd build
$ ../configure
$ make -j$(nproc) && make install
```

6.3 测试

```
$ cat main.c
#include <stdio.h>

int32_t op1 = 6;
int32_t op2 = 7;
int32_t res = 1;

int main(void)
{
    res += op1 * op2;
    printf("res = %d\n", res);
    return 0;
}
$ riscv64-unknown-elf-gcc –O2 main.c -o main.elf
$ riscv64-unknown-elf-objdump -D main.elf > main.asm
$ spike pk main.elf
```

查看反汇编文件 main.asm:

```
00000000000100b0 <main>:
   100b0:       7501a603                lw      a2,1872(gp) # 1f588 <op1>
   100b4:       7481a783                lw      a5,1864(gp) # 1f580 <res>
   100b8:       74c1a683                lw      a3,1868(gp) # 1f584 <op2>
   100bc:       6571                    lui     a0,0x1c
   100be:       1141                    addi    sp,sp,-16
   100c0:       aed6778b                mac     a5,a2,a3
   100c4:       65050513                addi    a0,a0,1616 # 1c650 <__clzdi2+0
   100c8:       e406                    sd      ra,8(sp)
   100ca:       0007859b                sext.w  a1,a5
   100ce:       74f1a423                sw      a5,1864(gp) # 1f580 <res>
   100d2:       200000ef                jal     ra,102d2 <printf>
   100d6:       60a2                    ld      ra,8(sp)
   100d8:       4501                    li      a0,0
   100da:       0141                    addi    sp,sp,16
   100dc:       8082                    ret
```

7. 总结

本文以在工具链中增加 **mac** 指令为例阐明整个开发流程，实质上只完成了其中最基础的部分。为了兼容标准的 RISC-V 指令集，我们还需要在 GCC 中增加如-march=rv64gc 的编译选项，用于控制是否使用 **mac** 指令，同时为了表明应用程序中包含了 **mac** 指令，还需要在 binutils 中设置一些标志位，声明程序使用到了扩展指令 **mac.**