



Imagination大学计划

RVfpga实验3

RISC-V汇编语言

1. 简介

对于程序员而言，C、Java和Python等高级语言能够提高编程的效率。这些高级语言被转换成汇编语言，即一组简单的指令。有时需要用汇编语言编写对性能或时序要求严格的代码段，以保证特定的时序或缩短计算时间。本实验将展示如何使用PlatformIO创建一个可在RVfpga系统上运行的RISC-V汇编语言程序。我们首先简要概述RISC-V汇编语言，然后展示如何在RVfpgaNexys上创建和运行汇编程序（请记住，也可以使用Verilator或Whisper仿真程序）。然后，我们会提供一些练习，让您编写自己的RISC-V汇编程序。

2. RISC-V汇编语言概述

RISC-V汇编语言包含用于实现高级代码的简单指令。例如add、sub和mul等常用的RISC-V指令（分别用于对两个操作数进行加法、减法和乘法运算）。

RISC-V指令的基本类型包括：计算（算术、逻辑和移位）指令、访存操作和分支/跳转。表1中列出了最常用的RISC-V指令。指令使用的操作数位于寄存器或存储器中，或者被编码为常量（即立即数）。RISC-V包含32个32位寄存器。表2列出了这32个RISC-V寄存器的名称。这些寄存器可以使用寄存器名称（例如zero、s0、t5等）或寄存器编号（即x0、x8、x30）来指定。程序员通常使用寄存器名称，其中保留有一些关于寄存器典型用途的信息。例如，保存寄存器s0-s11通常用于程序变量，而临时寄存器t0-t6用于临时计算。zero寄存器（x0）中始终包含值0，因为程序中通常需要该值。其他寄存器也均有特定用途，如表2所示，但在本实验中，只需使用zero寄存器、临时寄存器和保存寄存器。

表1. 常用的RISC-V汇编指令

	RISC-V汇编语言	说明	操作
计算	add s0, s1, s2	加法	s0 = s1 + s2
	sub s0, s1, s2	减法	s0 = s1 - s2
	addi t3, t1, -10	加立即数	t3 = t1 - 10
	mul t0, t2, t3	32位乘法	t0 = t2 * t3
	div s9, t5, t6	除法	t9 = t5 / t6
	rem s4, s1, s2	求余	s4 = s1 % s2
	and t0, t1, t2	按位与	t0 = t1 & t2
	or t0, t1, t5	按位或	t0 = t1 t5
	xor s3, s4, s5	按位异或	s3 = s4 ^ s5
	andi t1, t2, 0xFFB	按位与（立即数）	t1 = t2 & 0xFFFFFFF
	ori t0, t1, 0x2C	按位或（立即数）	t0 = t1 0x2C
	xori s3, s4, 0xABC	按位异或（立即数）	s3 = s4 ^ 0xFFFFFABC
	sll t0, t1, t2	逻辑左移	t0 = t1 << t2
	srl t0, t1, t5	逻辑右移	t0 = t1 >> t5
	sra s3, s4, s5	算术右移	s3 = s4 >>> s5
	slli t1, t2, 30	逻辑左移（立即数）	t1 = t2 << 30
	srli t0, t1, 5	逻辑右移（立即数）	t0 = t1 >> 5
	srai s3, s4, 31	算术右移（立即数）	s3 = s4 >>> 31

访存	lw s7, 0x2C(t1)	装载字	s7 = memory[t1+0x2C]
	lh s5, 0x5A(s3)	装载半字	s5 = SignExt(memory[s3+0x5A] _{15:0})
	lb s1, -3(t4)	装载字节	s1 = SignExt(memory[t4-3] _{7:0})
	sw t2, 0x7C(t1)	存储字	memory[t1+0x7C] = t2
	sh t3, 22(s3)	存储半字	memory[s3+22] _{15:0} = t3 _{15:0}
	sb t4, 5(s4)	存储字节	memory[s4+5] _{7:0} = t4 _{7:0}
分支	beq s1, s2, L1	如相等则分支跳转	if (s1==s2), PC = L1
	bne t3, t4, Loop	如不相等则分支跳转	if (s1!=s2), PC = Loop
	blt t4, t5, L3	如小于则分支跳转	if (t4 < t5), PC = L3
	bge s8, s9, Done	如大于或等于则分支跳转	if (s8>=s9), PC = Done
伪指令	li s1, 0xABCDEF12	装载立即数	s1 = 0xABCDEF12
	la s1, A	装载地址	s1 = 存储变量A的存储器地址
	nop	无操作	无操作
	mv s3, s7	移动	s3 = s7
	not t1, t2	非（求反）	t1 = ~t2
	neg s1, s3	求补	s1 = -s3
	j Label	跳转	PC = 标签
	jal L7	跳转并链接	PC = L7; ra = PC + 4
	jr s1	跳转寄存器	PC = s1

除了实际的RISC-V指令，RISC-V还包括伪指令（如表1底部所示），伪指令并非真正的RISC-V指令，但经常被程序员使用。伪指令是使用一个或多个真正的RISC-V指令实现的。例如，移动伪指令（mv s1, s2）会复制s2的内容并将其放入s1。该指令是使用以下真正的RISC-V指令实现的：addi s1, s2, 0。

表2. RISC-V寄存器

名称	寄存器编号	用途
zero	x0	常数值0
ra	x1	返回地址
sp	x2	堆栈指针
gp	x3	全局指针
tp	x4	线程指针
t0-2	x5-7	临时变量
s0/fp	x8	保存寄存器/帧指针
s1	x9	保存寄存器
a0-1	x10-11	函数参数/返回值
a2-7	x12-17	函数参数
s2-11	x18-27	保存寄存器
t3-6	x28-31	临时变量

以句点开头的命令为汇编器指令。这些指令是汇编器需执行的命令，而不是汇编器要转换的代码。这些指令会告知汇编器放置代码和数据的位置，指定在程序中使用的文本和数据常量等。表3列出了RISC-V的主要汇编器指令（《RISC-V读者：开源架构Atlas》，作者：Patterson & Waterman, © 2017）。

表3. RISC-V主要指令

指令	说明
.text	后续项目存储在text部分（机器代码）。
.data	后续项目存储在data部分（全局变量）。
.bss	后续项目存储在bss部分（全局变量初始化为0）。
.section .foo	后续项目存储在名为.foo的部分。
.align n	以2 ⁿ 字节边界对齐下一数据。例如，.align 2会以字边界对齐下一个值。
.balign n	以n字节边界对齐下一数据。例如，.balign 4会以字边界对齐下一个值。
.globl sym	声明标签sym为全局标签，可以从其他文件引用
.string "str"	将字符串str存储在存储器中并以空字符结尾。
.word w1,...,wn	将n个32位数值存储在连续的存储器字中。
.byte b1,...,bn	将n个8位数值存储在连续的存储器字节中。
.space	保留存储器空间以存储没有初始值的变量。该指令通常用于声明输出变量（这些输出变量不得同时作为输入变量）。要保留的空间必须始终以字节数形式表示。例如，指令RES: .space 4会保留四个未初始化的字节（即一个字）。
.equ name,constant	定义符号名称及其常量值。例如，.equ N,12会定义符号N，其值为12。
.end	汇编器在到达.end指令时会停止工作。该指令之后的所有文本都将被忽略。

以下示例（参见表4 - 表5）展示了如何以RISC-V汇编语言编码一些常见的高级结构。请注意，分支指令（beq、bne、blt和bge）会有条件地跳转到特定标签；而跳转指令（j）会无条件地跳转到特定标签。单行注释在C语言中以//表示，在RISC-V汇编语言中以#表示。

请注意，在第一个示例中（有关if/else语句的实现，请参见表4），C代码和RISC-V汇编代码会检查相反的情况：C代码检查小于（<）某值的情况，而相应的汇编代码检查大于或等于（>=）某值的情况。

表4. RISC-V汇编示例1: if/else语句

// C Code	# RISC-V Assembly
int a, b, c;	# s0 = a, s1 = b, s2 = c
if (a < b)	bge s0, s1, L1 # if (a >= b) goto L1
c = 5;	addi s2, zero, 5 # c = 5
else	j L2 # jump over else block
c = a + b;	L1: add s2, s0, s1 # c = a + b
	L2:

在第二个示例中（有关整数数组的操作，请参见表5），RISC-V汇编代码使用临时寄存器（t0-t3）来保存临时值，例如常量100和数据数组的基址。对前三个指令中的寄存器进行初始化后，RISC-V汇编代码会使用**bge**（如大于或等于则分支）指令检查*i* >= 100的情况；这同样与C代码相反。如果满足该条件，则完成for循环。如果未进行分支，则*i*小于100，将执行其余代码。请注意，索引*i*要先乘以4（使用slli t2, s0, 2指令）再添加到基址中，因为整数（32位二进制补码）会占用4个字节的存储空间。在RISC-V中，存储空间可按字节寻址（即每个字节都有自己的地址）。如果数组为字符数组（即char data[100];），每个数组元素将只占用一个字节，并且可以直接将*i*添加到基址中，形成数组索引*i*的地址，即array[i]。读取数组元素后，会将其减10，然后将其写入（分别通过lw、addi和sw指令），数组索引*i*（即s0）将递增，程序跳回到for循环的开头（使用j L5指令）。

表5. RISC-V汇编示例2：操作整数数组

// C Code	# RISC-V Assembly
int i;	# s0 = i, t1 = base address of data (assumed
int data[100];	# to be at 0x300)
for (i=0; i<100; i++)	addi s0, zero, 0 # i = 0
	addi t0, zero, 100 # t0 = 100
	li t1, 0x300 # base address of array
	L5: bge s0, t0, L7 # if (i>=100) exit loop
	slli t2, s0, 2 # t2 = i*4
	add t2, t1, t2 # address of data[i]
	lw t3, 0(t2) # t3 = array[i]
array[i] = array[i]-10;	addi t3, t3, -10 # t3 = array[i]-10
	sw t3, 0(t2) # array[i] = array[i]-10
	addi s0, s0, 1 # i++
	j L5 # loop
	L7:

有关RISC-V汇编语言的更多详细信息，请参见《RISC-V指令集手册》（下载地址：<https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf>）或《数字设计和计算机体系结构》（作者：Harris & Harris, Elsevier, © 2021，出版时间：2021年夏季）或《RISC-V读者：开源架构Atlas》（作者：Patterson & Waterman, © 2017）。

3. 为RVfpga编写RISC-V汇编程序

现在您可以着手编写自己的RISC-V汇编程序。在自行编写程序之前，请按照以下步骤设置PlatformIO项目，然后在RVfpgaNexys上创建并运行汇编程序（请记住，也可以使用Verilator或Whisper仿真这些程序）：

1. 创建RVfpga项目
2. 编写RISC-V汇编语言程序
3. 将RVfpgaNexys下载到Nexys A7 FPGA开发板
4. 编译、下载和运行汇编程序

步骤1. 创建RVfpga项目

按RVfpga实验2中的第1步进行操作 - 为方便起见，下文将重述相关操作。单击“Start”（开始）按钮，输入“VSCode”，然后单击Visual Studio Code以打开VSCode（参见图1）。

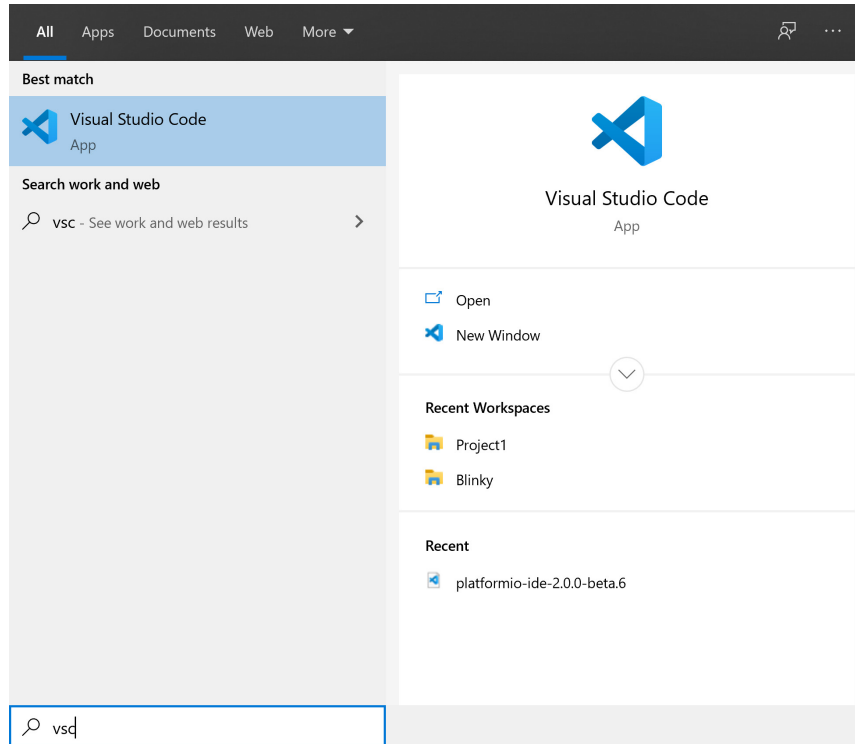


图1. 打开VSCode

如果启动VSCode时PlatformIO没有自动打开，请单击左侧功能区菜单中的PlatformIO图标，然后单击“PIO Home”（PIO主页）→“Open”（打开）（参见图2）。

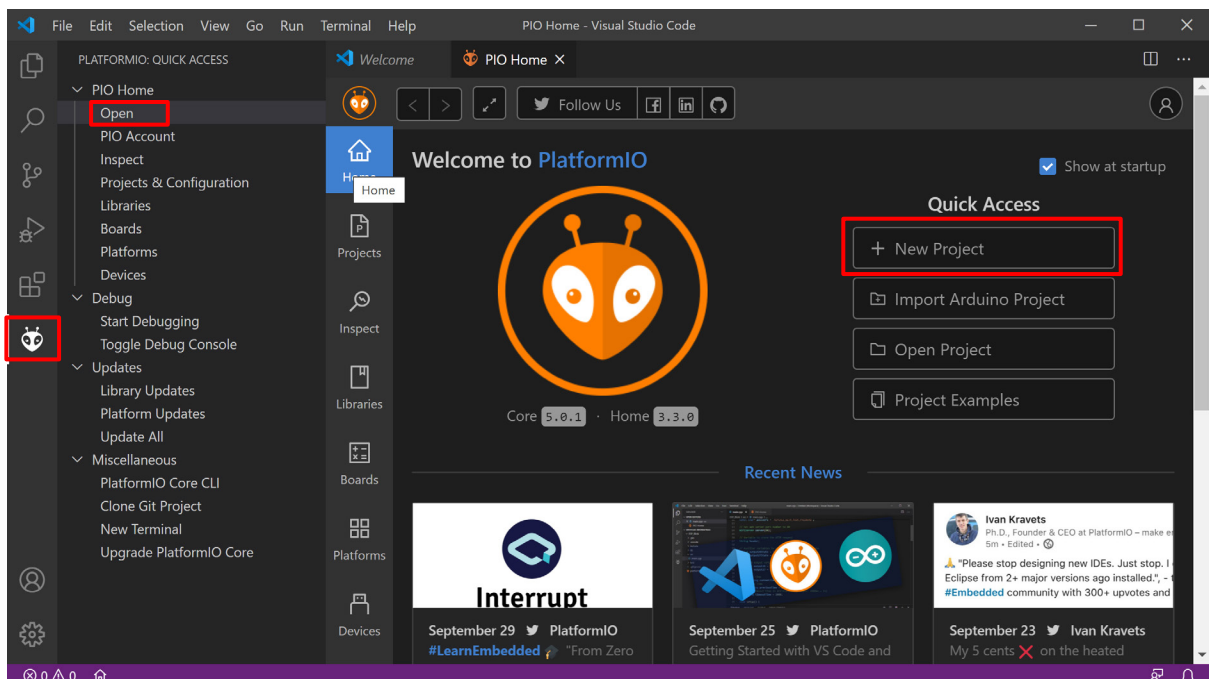


图2. 打开PlatformIO并创建新项目

现在，在“PIO Home”（PIO主页）欢迎窗口中，单击“New Project”（新建项目）（参见图2）。

如图3所示，将项目命名为“Project1”，在“Board”（开发板）部分选择“RVfpga: Digilent Nexys A7”（输入RVfpga，即可出现该选项）。保留框架的默认选项WD-framework（Western Digital框架，其中包含Freedom-E SDK gcc和gdb）。取消选中“Use default location”（使用默认位置），然后将程序放到以下路径：

```
[RVfpgaPath]/RVfpga/Labs/Lab3
```

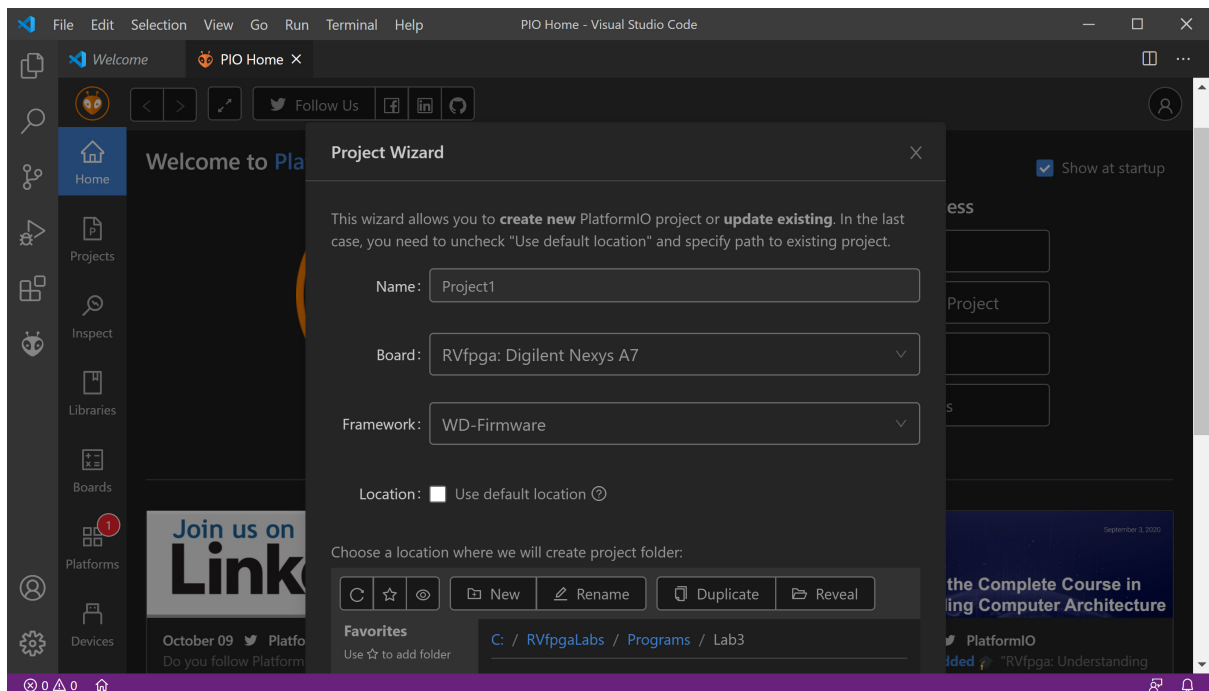


图3. 为项目命名并选择开发板和项目文件夹

然后点击窗口底部的“Finish”（完成）（参见图4）。

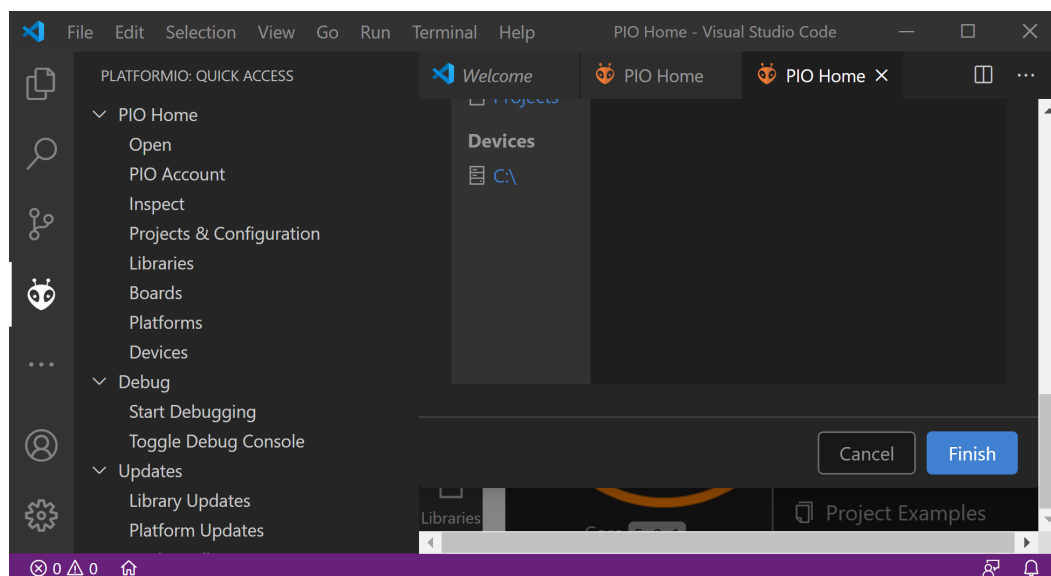


图4. 完成项目创建

在左侧“Explorer”（资源管理器）窗格的PROJECT1下（可能需要展开），双击platformio.ini打开该文件（参见图5）。该文件为PlatformIO初始化文件。

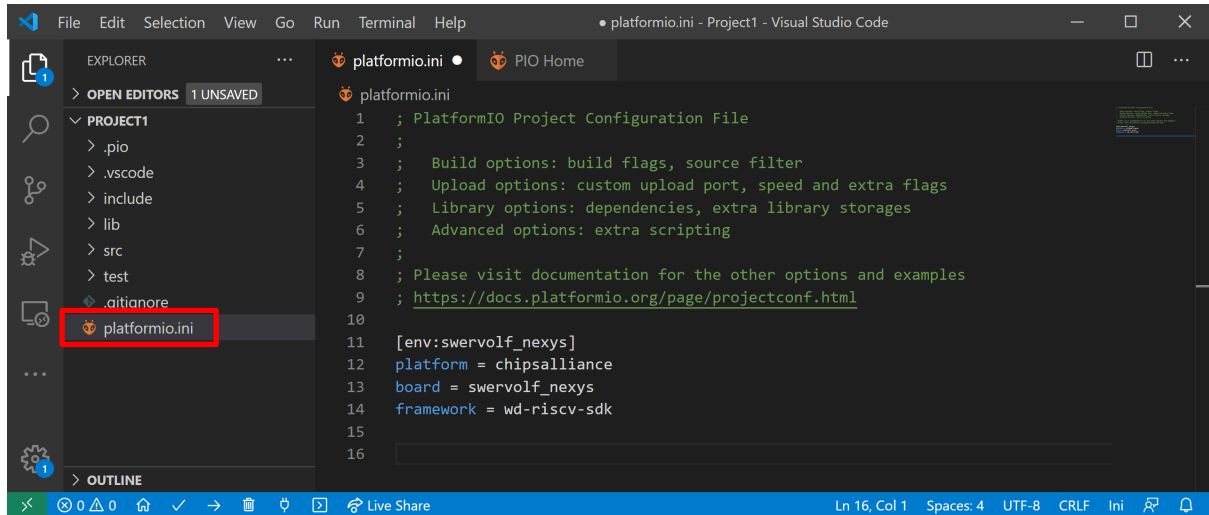


图5. PlatformIO初始化文件：platformio.ini

将以下代码行添加到platformio.ini文件，如图6所示：

```
board_build.bitstream_file =
[RVfpgaPath]/RVfpga/Labs/Lab1/Project1/Project1.runs/impl_1/rvfpganexys.bit
```

此行代码指示PlatformIO可以在哪里找到要加载到FPGA上的比特流文件。上述路径是在实验1中创建的比特流的位置。（如果尚未完成实验1，可以使用“入门指南”中随附的RVfpgaNexys比特流，路径为：`[RVfpgaPath]/RVfpga/src/rvfpganexys.bit`。）按Ctrl-s保存platformio.ini文件。

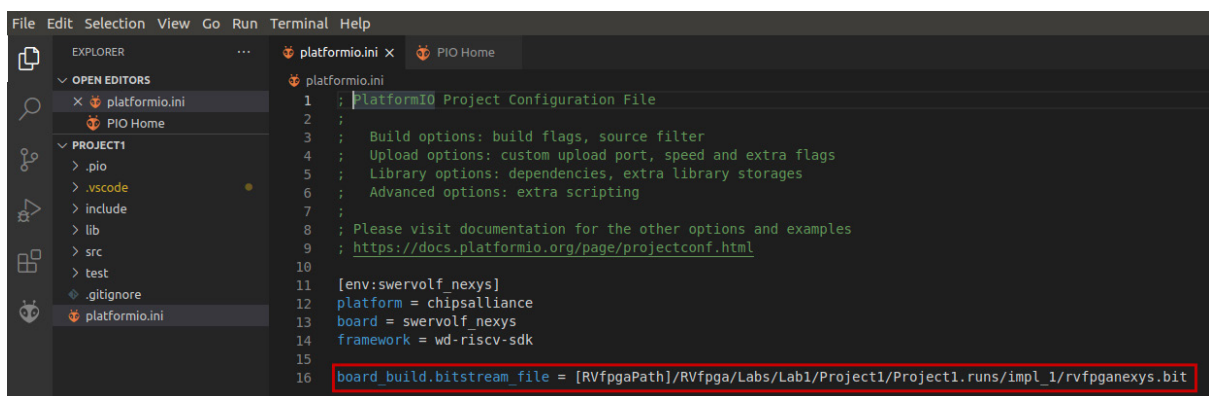


图6. 添加RVfpgaNexys比特流文件（rvfpganexys.bit）的位置

请记住，“入门指南”的示例中使用了更完整的platformio.ini文件。如果要使用任何需要额外命令的功能（例如Verilator仿真器的路径、串行控制台的配置、Whisper调试工具等），则可以使用上述示例中的platformio.ini。

步骤2. 编写RISC-V汇编语言程序

现在，需要编写RISC-V汇编程序。单击“File”（文件）→“New File”（新建文件）（参见图7）。

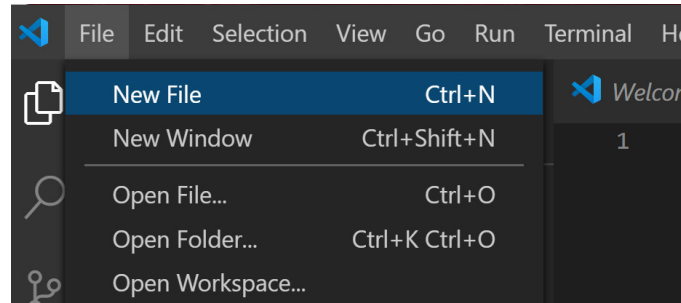


图7. 向项目中添加文件

将打开一个空白窗口。在该窗口中输入（或复制/粘贴）以下RISC-V汇编程序（参见图8）。也可通过以下路径获取该程序：

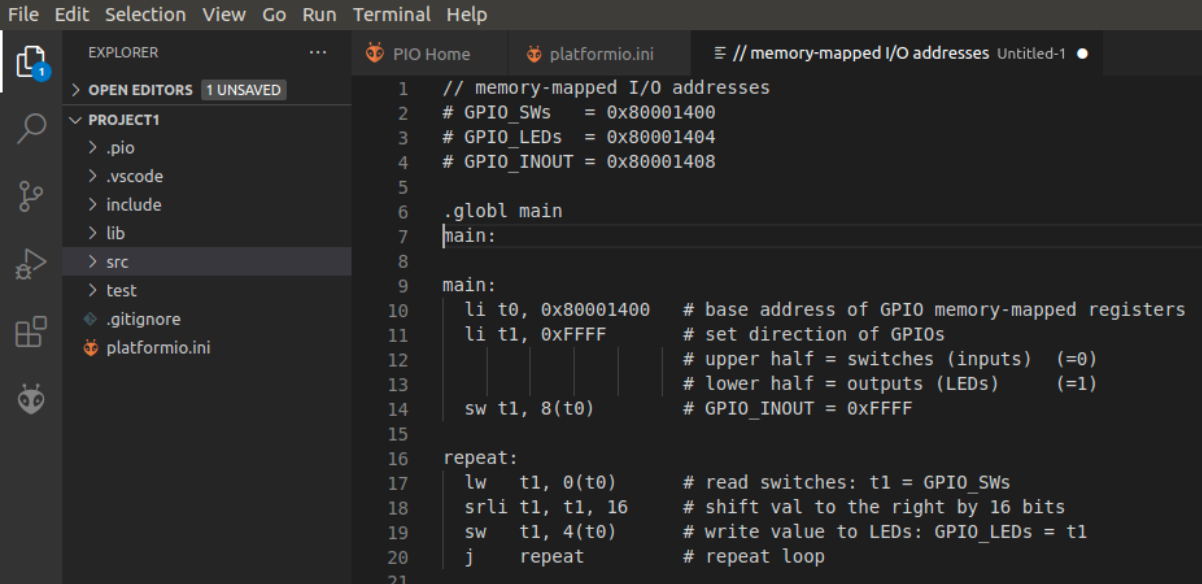
[RVfpgaPath]/RVfpga/Labs/Lab3/ReadSwitches.S

```
// memory-mapped I/O addresses
# GPIO_SWs   = 0x80001400
# GPIO_LEDs  = 0x80001404
# GPIO_INOUT = 0x80001408

.globl main
main:

main:
    li t0, 0x80001400    # base address of GPIO memory-mapped registers
    li t1, 0xFFFF       # set direction of GPIOs
                        # upper half = switches (inputs)   (=0)
                        # lower half = outputs (LEDs)      (=1)
    sw t1, 8(t0)         # GPIO_INOUT = 0xFFFF

repeat:
    lw  t1, 0(t0)        # read switches: t1 = GPIO_SWs
    srli t1, t1, 16       # shift val to the right by 16 bits
    sw  t1, 4(t0)        # write value to LEDs: GPIO_LEDs = t1
    j   repeat           # repeat loop
```



```

1 // memory-mapped I/O addresses
2 # GPIO_SWs = 0x80001400
3 # GPIO_LEDs = 0x80001404
4 # GPIO_INOUT = 0x80001408
5
6 .globl main
7 main:
8
9 main:
10     li t0, 0x80001400 # base address of GPIO memory-mapped registers
11     li t1, 0xFFFF    # set direction of GPIOs
12                     # upper half = switches (inputs) (=0)
13                     # lower half = outputs (LEDs) (=1)
14     sw t1, 8(t0)      # GPIO_INOUT = 0xFFFF
15
16 repeat:
17     lw t1, 0(t0)      # read switches: t1 = GPIO_SWs
18     srli t1, t1, 16    # shift val to the right by 16 bits
19     sw t1, 4(t0)      # write value to LEDs: GPIO_LEDs = t1
20     j repeat          # repeat loop
21

```

图8. 输入RISC-V汇编程序

汇编代码的开头必须包含以下代码行：

```
.globl main
main:
```

汇编器指令 `.globl` 会使标签在所有链接文件中可见。引导代码

（`~/platformio/packages/framework-wd-riscv-sdk/board/nexys_a7_eh1/startup.S`）将配置系统并跳转到该标签（`main`）。调试器将在开始工作时在该处设置一个临时断点。

此RISC-V汇编程序与实验2中的示例程序相同，但这次是用RISC-V汇编语言编写的。此程序会设置通用I/O（GPIO）的输入和输出方向，然后反复读取开关的值并将该值写入LED。

将程序输入窗格后，按 **Ctrl-s** 保存文件。将其命名为 `ReadSwitches.S`，并保存到 `Project1` 目录下的 `src` 文件夹中（参见图9）。

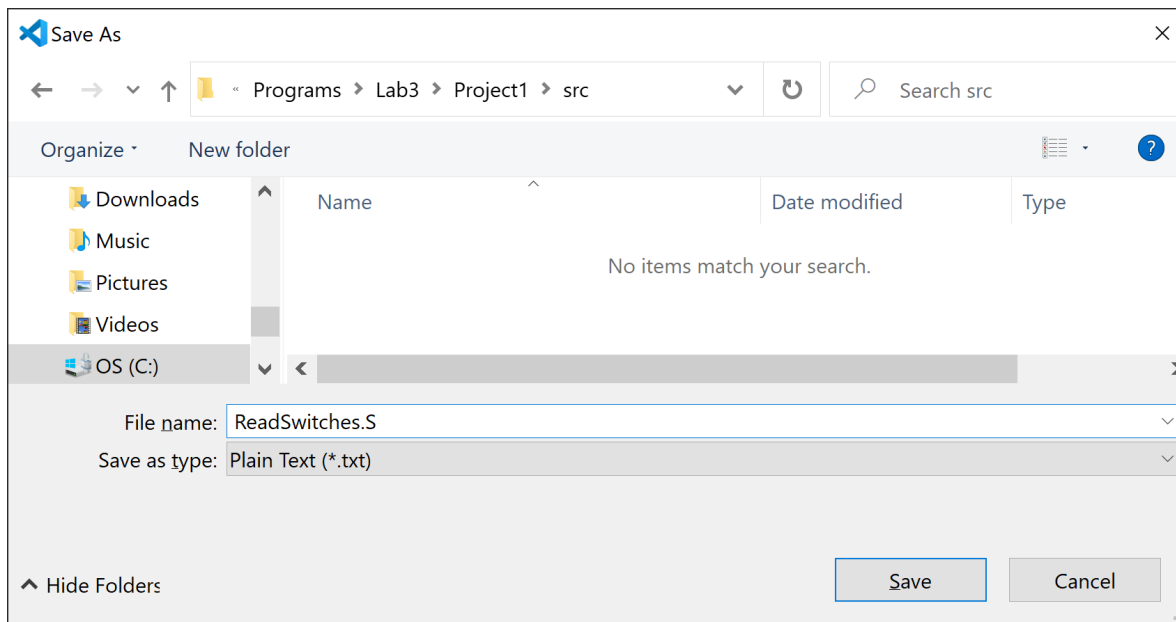




图9. 将文件另存为ReadSwitches.S

步骤3. 将RVfpgaNexys下载到Nexys A7 FPGA开发板

现在，需要将RVfpgaNexys下载到Nexys A7 FPGA开发板上。根据GSG和实验2中的指示下载RVfpgaNexys - 为方便起见，下文将重述相关操作。

单击左侧功能区菜单中的PlatformIO图标，展开“Project Tasks”（项目任务）→ env:swervolf_nexys → “Platform”（平台），然后单击“Upload Bitstream”（上传比特流），将RVfpgaNexys下载到Nexys A7开发板。

也可以使用PlatformIO终端窗口下载RVfpgaNexys，方法是单击PlatformIO窗口底部菜单中的“PlatformIO: New Terminal”（PlatformIO: 新建终端）按钮，然后将以下内容输入（或复制粘贴）到PlatformIO终端中：

```
pio run -t program_fpga
```

步骤4. 编译、下载和运行RISC-V汇编程序

现在RVfpgaNexys已在开发板上运行，接下来需要编译程序，将程序下载到RVfpgaNexys，然后运行/调试程序。如果VSCode尚未打开，请将其打开。上一个项目Project1应该会自动打开。如果该项目未自动打开，请确保已打开PlatformIO扩展，然后单击“File”（文件）→ “Open Folder”（打开文件夹）并选择（但不要打开）在本实验前面部分创建的Project1。

单击左侧功能区菜单中的“Run”（运行）按钮，然后单击“Start Debugging”（开始调试）按钮（参见图10）。

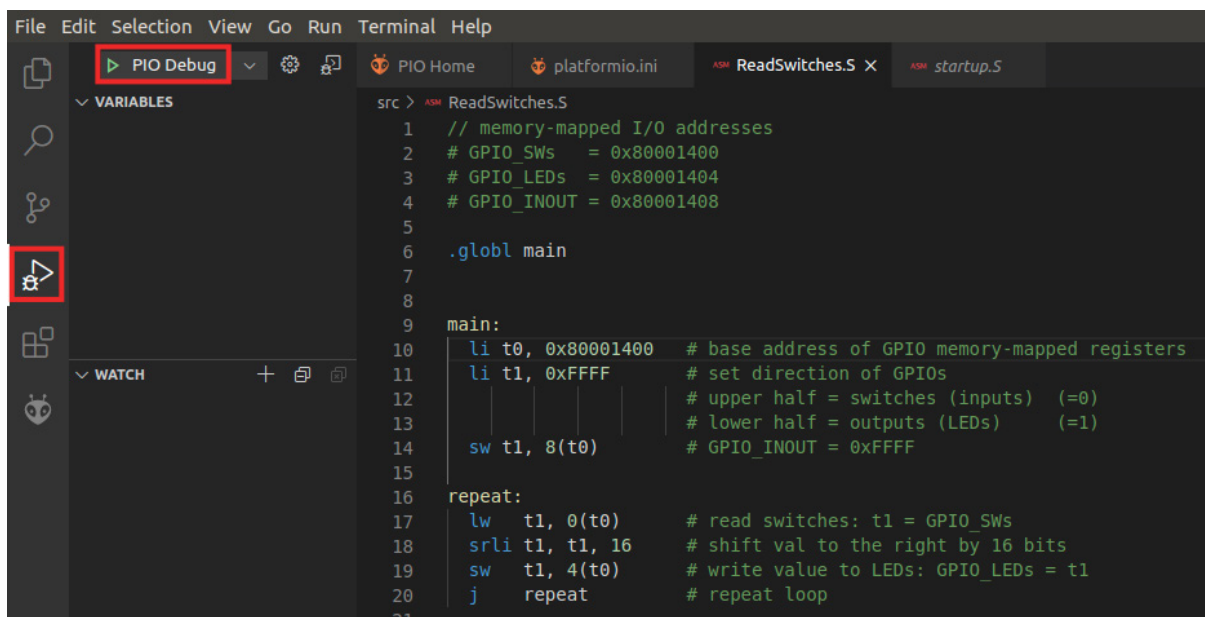


图10. 在RVfpgaNexys上运行程序

程序将下载至正在Nexys A7FPGA开发板上运行的RVfpgaNexys中。此时即可开始运行和调试程序（参见图11）。

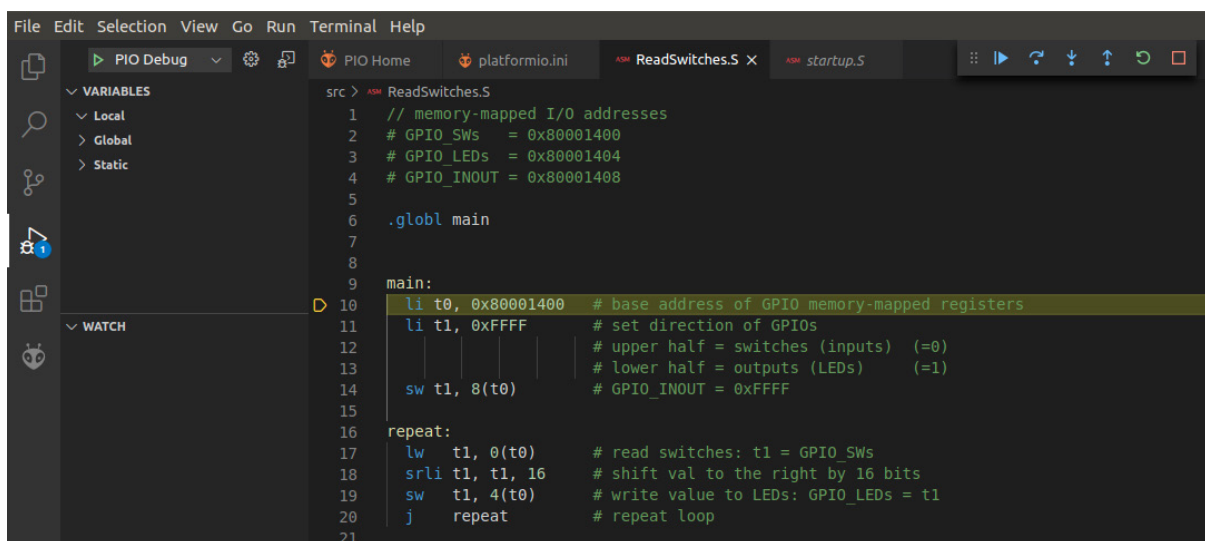



图11. 在RVfpgaNexys上运行的程序

根据“RVfpga入门指南”和实验2中的指示，使用调试工具栏和调试器选项运行和管理程序。例如，可以在第17行设置一个断点（只需单击行号左侧），然后查看寄存器t1，因为开关的值已加载到寄存器中。按下“Stop”（停止）按钮  （或Shift - F5）停止调试会话后，调试会话将终止，但程序仍继续在RVfpgaNexys上运行。

4. 练习

现在可再次重复实验2中的练习，将C语言替换为RISC-V汇编语言，创建自己的RISC-V汇编程序。为方便起见，下文再次提供了练习说明。

请记住，如果使Nexys A7开发板与计算机保持连接并保持上电状态，则在切换运行不同程序时，无需将RVfpgaNexys重新加载到开发板上。但是，如果关闭Nexys A7开发板，则需要使用PlatformIO将RVfpgaNexys重新加载到开发板上。

还要记住，可以使用Verilator或Whisper仿真这些程序。

练习1. 编写一个RISC-V汇编程序，使开关的值在LED上闪烁。该值应以足够慢的速度进行脉冲开关，让人眼可以观察到值在闪烁。将程序命名为**FlashSwitchesToLEDs.S**。

练习2. 编写一个RISC-V汇编程序，在LED上显示开关值取反后的值。例如，如果开关的值（二进制）为：0101010101010101，则LED应显示：1010101010101010；如果开关的值为：1111000011110000，则LED应显示：0000111100001111；依此类推。将程序命名为**DisplayInverse.S**。

练习3. 编写一个RISC-V汇编程序，不断增加点亮并滚动的LED的数量，直到所有LED都点亮为止。然后该模式应重复进行。将程序命名为**ScrollLEDs.S**。

该程序应产生以下效果：

1. 首先，有一个点亮的LED从右向左滚动。
2. 到达最左侧的LED后，应有两个点亮的LED从左向右滚动，然后又从右向左滚动。
3. 到达最左侧的两个LED后，应有三个点亮的LED从左向右滚动，然后又从右向左滚动。
4. 接下来应有四个点亮的LED滚动。
5. 依此类推，直到所有LED均点亮。
6. 然后该模式应重复进行。

练习4. 编写一个RISC-V汇编程序，显示开关的4个最低有效位和开关的4个最高有效位相加所得的4位无符号值。在LED的4个最低有效（最右边）位上显示结果。将程序命名为**4bitAdd.S**。当发生无符号溢出（即进位为1）时，LED的第五位应亮起。

练习5. 编写一个RISC-V汇编程序，根据欧几里得算法求出a和b两个数的最大公约数。a和b两个值在程序中应该是静态定义的变量。将程序命名为**GCD.S**。有关欧几里得算法的更多信息，请访问：<https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/the-euclidean-algorithm>。也可以直接用google搜索“欧几里德算法”。

练习6. 编写一个RISC-V汇编程序，计算斐波那契数列中的前12个数，并将结果存储在长度为12的有限向量（即数组）**V**中。这个无限的斐波纳契数列定义如下：

$$V(0)=0, V(1)=1, V(i)=V(i-1)+V(i-2) \text{ (其中 } i=0,1,2,\dots)$$

换言之，与元素*i*对应的斐波那契数是数列中在其之前的两个斐波那契数之和。表6显示了*i*为0到8时的斐波那契数。

表6. 斐波那契数列

<i>i</i>	0	1	2	3	4	5	6	7	8
V	0	1	1	2	3	5	8	13	21

向量的维数*N*在程序中必须定义为常量。将程序命名为**Fibonacci.S**。

练习7. 现有一个*N*维向量（即数组）**A**，请生成另一个向量**B**，使向量**B**只包含**A**中大于0的偶数元素。例如：假设*N* = 12, **A** = [0,1,2,7,-8,4,5,12,11,-2,6,3]，则**B**为：**B** = [2,4,12,6].将程序命名为**EvenPositiveNumbers.S**。

练习8. 现有两个*N*维向量（即数组）**A**和**B**，请生成另一个向量**C**，定义如下：

$$C(i) = |A[i] + B[N-i-1]|, i = 0, \dots, N-1.$$

用RISC-V汇编语言编写一个计算新向量的程序。在程序中使用包含12个元素的数组。将程序命名为**AddVectors.S**。

练习9. 用RISC-V汇编语言实现冒泡排序算法，让算法通过以下方式按升序对向量分量进行排序：

1. 反复遍历向量直至完成。
2. 如果两个相邻分量 $V(i) > V(i+1)$ ，则互换其位置。
3. 当每对相邻分量的顺序均正确时，算法停止。

使用包含12个元素的数组测试程序。将程序命名为**BubbleSort.S**。

练习10. 用RISC-V汇编语言编写一个程序，通过迭代乘法计算给定非负数*n*的阶乘。虽然应使用多个*n*值测试程序，但最终提交的应为*n* = 7时的结果。*n*应为程序内静态定义的变量。将程序命名为**Factorial.S**。