|  |  |  |
| --- | --- | --- |
| Troubleshooting  **This course has not yet been updated to work with the Raspberry Pi models B+ and A+. Some elements may not work, in particular the first few lessons about the LED. It has also not been updated for Raspberry Pi v2.**  There is little more satisfying than when an Operating System you've written works perfectly, however this is, unfortunately, rarely the case. I lost count long ago how many Operating Systems I've written that didn't work in making this course. This page contains advice for what to do when things just aren't working. It is broken down into compile errors that happen before you can even make the Operating System, load errors that prevent your Operating System doing anything, and runtime errors, where your Operating System doesn't do the correct thing. I've also added specific help on each of the tutorial Operating Systems, such as things that can commonly go wrong. If you have a problem that isn't explained here, and think others my be able to benefit from your experience, send an email to me at [awc32@cam.ac.uk](mailto:awc32@cam.ca.uk), and I will add it here. It is also well worth taking a look at the [Raspberry Pi forums](http://www.raspberrypi.org/phpBB3/viewforum.php?f=72) to see if anyone else has run into the same problem.   |  | | --- | | **Contents**   * [1 Compiler Errors](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/troubleshooting.html#compile)   + [1.1 Error: bad instruction](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/troubleshooting.html#badinstruction)   + [1.2 Error: immediate expression requires a # prefix](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/troubleshooting.html#immediate)   + [1.3 Error: ARM register expected](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/troubleshooting.html#registerexpected)   + [1.4 Error: unknown pseudo-op](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/troubleshooting.html#pseudoop)   + [1.5 Error: invalid constant (number) after fixup](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/troubleshooting.html#invalidconstant)   + [1.6 warning : end of file not at end of a line; newline inserted](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/troubleshooting.html#newline)   + [1.7 undefined reference](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/troubleshooting.html#undefinedref)   + [1.8 `section' referenced in section `section' of build/file.o: defined in discarded section `section' of build/file.o](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/troubleshooting.html#sections)   + [1.9 arm-none-eabi-ld: no input files](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/troubleshooting.html#nofiles)   + [1.10 make: \*\*\* No targets specified and no makefile found. Stop.](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/troubleshooting.html#nomake)   + [1.11 Windows Only: make: Interrupt/Exception caught (code = 0xc00000fd, addr = 0x425073)](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/troubleshooting.html#win64)   + [1.12 Linux 64 bit Only: arm-none-eabi-as: No such file or directory](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/troubleshooting.html#linux64) * [2 Load Errors](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/troubleshooting.html#load) * [3 Runtime Errors](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/troubleshooting.html#runtime)   + [3.1 Alignment](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/troubleshooting.html#wordalign)   + [3.2 Hanging](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/troubleshooting.html#hang)   + [3.3 Infinite Loops](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/troubleshooting.html#infinity) * [4 General Advice](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/troubleshooting.html#advice) * [5 Tutorial Specific Advice](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/troubleshooting.html#tutorials)   + [5.1 OK05 Doesn't Flash; Light Stays On](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/troubleshooting.html#ok05stuck)   + [5.2 Screen01 Displays Nothing](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/troubleshooting.html#screen01blank)   + [5.3 Screen02 Displays Nothing](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/troubleshooting.html#screen02blank)   + [5.4 Input01 Displays Nothing](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/troubleshooting.html#input01blank) |   1 Compiler Errors  Compiler errors are errors that occur when the **make** command runs on your Operating System. I've also included some common warnings too.  **1.1 Error: bad instruction**  source/file.s:8: Error: bad instruction `sdd r0,r1'  This error occurs when you use a command that doesn't exist. First of all, check that you haven't mistyped the command. If you're using condition codes as well as other options such as an **str** command with a **b** suffix to only store a byte, and a **eq** suffix to only store if the last condition was equal, the correct order is in fact **streqb** not **strbeq**.  **1.2 Error: immediate expression requires a # prefix**  source/file.s:32: Error: immediate expression requires a # prefix -- `add r0,1'  This means that you're trying to use a constant number, such as adding the number one, but forgot to put a # (e.g. **add r3,4** should be **add r3,#4**. You must do so wherever you use a constant on a command that normally uses registers. This is true even for calculated constants such as **#3\*4**.  **1.3 Error: ARM register expected**  source/file.s:24: Error: ARM register expected -- `add 0,r1'  This means that you typed something that was not a register, when a register was expected. Double check your spelling, especially if you're using **.req**. If you are, make sure you haven't used **.unreq** between the **.req** and this command.  **1.4 Error: unknown pseudo-op**  source/file.s: Error: unknown pseudo-op: `.suction'  This error occurs when you use a pseudo operation that doesn't exit. Check your spelling.  **1.5 Error: invalid constant (number) after fixup**  source/file.s:24: Error: invalid constant (c21) after fixup  This error occurs when you use a constant which does not meet the requirements of the function. The most common example of this is the **mov** instruction, which only allows numbers which can be represented as an 8 bit number, shifted left by an even number. For example c2116 = 1100001000012 and so cannot be represented in a **mov**, but c2016 = 1100001000002 = 110000102 << 4, and so is valid in a **mov**. Much the same rules apply to most constants in functions. Remember, to load in any constant, use **ldr r0,=value**.  **1.6 warning : end of file not at end of a line; newline inserted**  source/file.s: warning : end of file not at end of a line; newline inserted  This means the last line in your file is not empty. You can ignore this, but to fix it just add a new line at the end.  **1.7 undefined reference**  .text(+0x18): undefined reference to `label'  This means you've used a label which the linker can't find. This is probably due to a misspelling. Remember that labels are case sensitive and that labels in different files require **.globl** commands before they're accessible in other files.  **1.8 `section' referenced in section `section' of build/file.o: defined in discarded section `section' of build/file.o**  `.trxt' referenced in section `.init' of build/main.o: defined in discarded section `.trxt' of build/main.o  This means that you've used a **.section** command, but you've specified a section other than .init, .text or .data. Only these sections are copied into the kernel.img file, any others are discarded, hence the error is saying that some of your code was discarded. Check your spelling on **.section** commands.  **1.9 arm-none-eabi-ld: no input files**  arm-none-eabi-ld: no input files  This error means that the linker hasn't found your code. Double check you've got a source directory with .s files within it like main.s. Make sure you haven't got something like main.s.txt.  **1.10 make: \*\*\* No targets specified and no makefile found. Stop.**  make: \*\*\* No targets specified and no makefile found. Stop.  This error is caused by running make in the wrong directory. The command line must have the same working directory as the file makefile which is in the template. Use the 'cd' command to change directories, then run make again.  **1.11 Windows Only: make: Interrupt/Exception caught (code = 0xc00000fd, addr = 0x425073)**  make: Interrupt/Exception caught (code = 0xc00000fd, addr = 0x425073)  This is an error that can occur on Windows when YARGTO has been installed in a directory with a space in it's name, for example: C:\Program Files (x86)\YAGARTO\. To fix, please reinstall YAGARTO in a directory with no spaces such as C:\YAGARTO\.  **1.12 Linux 64 bit Only: arm-none-eabi-as: No such file or directory**  bash: arm-none-eabi-as: No such file or directory  This error is caused by running the Linux version of the toolchain on a 64 bit machine without 32 bit compatibility libraries. These can be retrieved easily using:  **sudo apt-get install ia32-libs**  2 Load Errors  Load errors are errors that occur that prevent your Operating System from giving any output. This can be the hardest to diagnose and fix. Unfortunately, by their nature, they give off no indication of what is wrong.  The first thing you should check is that the tutorial answer does work. This confirms that you're installing things correctly, that your Raspberry Pi is not physically damaged and that your SD card works. If the answer doesn't work, make sure Linux still does. If it doesn't you may have a problem with your SD card or Raspberry Pi physically. Reimage the SD card or get a new one. If Linux does work but the tutorial does not, you may not be installing the Operating System correctly. Double check you're replacing kernel.img in the FAT partition of the SD card.  If the answer does work but your attempt does not, then we know it is something in your code. On the later tutorials, try altering the start of your code to turn on the OK LED, just so you know if it boots at all. If not, double check that you have some code in the .init section which branches into your .text section. Try enabling the OK LED from the .init section.  Ultimately what we need is some output. If you can get the LED to turn on from your early code, then this is just a runtime error. If placing that code in the .init section still doesn't enable output, it may be worth going back to the template and copying in your code bit by bit until it stops working. Sometimes you never do find the error; In the past I've ended up copying the entire code back into the template and suddenly it worked.  3 Runtime Errors  Behind load errors, runtime errors are the hardest to diagnose and fix. These occur when your Operating System just doesn't do what you want.  The most important thing is to get information out of the system. The OK LED is very useful for this. If the Operating System seems to stop, or get stuck, try turning on the LED just before and just after various commands. If it turns on when just before an instruction, but won't on the instruction afterwards, then we know this is the problem. Please remember that turning on the LED will generally alter r0 to r3, so use **push {r0,r1,r2,r3}** and **pop {r0,r1,r2,r3}** to preserve these registers. If you're in looping code, try flashing the LED in the loop to see how many loops actually happen.  If you're in the later tutorials make sure to use the screen for information. Write out text about the current status, values, etc, in order to learn what is going on. Once you've got some idea, have a look at the following common problems to see if you can spot what has happened.  Remember, think outside the box. Perhaps a function you wrote ages ago had a bug you never noticed.  **3.1 Alignment**  One of the most subtle runtime errors is the ARM alignment constraint. Any **str** or **ldr** instruction will not function correctly unless the computed address is a multiple of the size of the data being read. For example, if you're using **ldr r0,[r1,#2]**, then the value of **r1,#2** must be a multiple of 4. If not unpredictable results occur. You should always be able to guarantee this is this case. If you're referring to a label, make sure you have a **.align 2** command BEFORE the label. This will ensure that the label's address is a multiple of 22 = 4. You can use **.align 3** to align to a multiple of 23 = 8, etc.  **3.2 Hanging**  A processor 'hangs' (stops) if it encounters a bad instruction or a bad address. If your code gets stuck on a branch, load or store command, this is likely to be the problem. You can use a condition around turning on the OK LED to check this.  **3.3 Infinite Loops**  Similarly to hanging, a processor can easily get stuck in loops. If the processor reaches one of your loops, but never leaves, this could well be the problem. Double check the conditions for leaving the loop will be satisfied.  4 General Advice  To help you, every time you compile a kernel, two extra files are compiled. kernel.list contains a direct listing of all the assembly code, and kernel.map contains a map of all your labels. You can use these files to mentally simulate the processor and check it will do the correct thing. The processor starts at address 0 with all registers in an undefined state. Try mentally checking that the processor will do what you want. For things like alignment issues, you can double check everything is as it should be with kernel.map.  5 Tutorial Specific Advice  **5.1 OK05 Doesn't Flash; Light Stays On**  If you followed a previous version of this tutorial, a common problem on OK05 is to have the OK LED stay on continuously rather than flashing a pattern. This is caused by a change in the way the modern bootloaders load the kernel; they load it at address 0x8000 not 0. Either replace makefile and kernel.ld with the ones currently in the template. Or alternatively add the line **kernel\_old=1** to the file config.txt on the SD card, or create the file with this line in it.  **5.2 Screen01 Displays Nothing**  If you followed a previous version of this tutorial, this was a common problem. The code in framebuffer.s has been altered to fix this problem. Specifically, it is necessary to add 0x40000000 to the address of FrameBufferInfo before writing it to the mailbox.  **5.3 Screen02 Displays Nothing**  See [OK05 Doesn't Flash; Light Stays On](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/troubleshooting.html#ok05stuck).  **5.4 Input01 Displays Nothing**  First of all, check if this is an issue with the screen or the keyboard by running the solution to Input02. It prints a message to the screen before receiving keyboard input. If nothing shows on Input02, then see the help for [Screen01](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/troubleshooting.html#screen01blank). If it does display, but you still can't type, then your keyboard may be incompatible with my USB driver. Unfortunately, due to it's basic code, the driver doesn't support every keyboard. Try to find other USB keyboards to use. I've personally tested 11 brands of keyboard, of which 6 worked. | 问题解决页  本课程还没有更新到可以在树莓派版本B+和A+。一些部分可能无法工作，尤其是可能是开头的一些和LED相关的课程。而且，本课程还没有更新到树莓派第二版。  没有什么比写出一个可以完美运行的操作系统更加令人满意的事情了。然而，不幸的是本课程并不是令人惬意。在写这个操作系统的过程中，我已经不太记得有多少次失败了。这个页面包含一些建议，当你遇到麻烦时，可以来参考参考。这个页面分为编译错误，它们出现在当你准备编译操作系统之前；加载错误，它将会组织操作系统去做任何事情；运行时错误，这暗示着操作系统没有做正确的事情。我会给每个操作系统教程都附加上一些特殊的帮助，来提示哪些经常容易犯的错误。如果你遇到了一个我们没有提及的问题，你可以考虑自己动手解决，并给发送邮件到我的邮箱：[bostonhsu@outlook.com](mailto:bostonhsu@outlook.com)，我会把它们罗列在这里。如果遇到类似的问题，到树莓派的论坛上找找其他人的答案还是很值得的。  目录  1. 编译错误  1.1 错误：错误指令  1.2 错误：直接表达式需要一个#前缀  1.3 错误：期望ARM寄存器  1.4 错误：未知的伪操作  1.5 错误：修复之后的无效常数（数值）  1.6 警告：文件的结尾并不是文件内容的最后一行；新行插入  1.7 没有定义的引用  1.8 段内参考了build/file.o段：定义在丢弃的build/file.o段  1.9 arm-none-eabi-ld:没有输入文件  1.10 make:\*\*\* 目标不明确并且makefile文件没有找到。终止。  1.11 只在Windows操作系统中出现：make:Interrupt/Exception caught(code = 0xc000000fd, addr = 0x425073)  1.12 只在64位Linux操作系统出现：arm-none-eabi-as: No such file or directory  2. 加载错误  3. 运行时错误  3.1 对齐  3.2 挂起  3.3 无穷循环  4. 一般建议  5. 辅导特别建议  5.1 OK05 不要闪烁；让灯保持开启状态  5.2 Screen01 什么也不显示  5.3 Screen02 什么也不显示  5.4 Input01 什么也不显示  1. 编译错误  编译错误是那些发生当make命令在你的操作系统中运行时发生的错误。我把一些警告信息也囊括在内。  1.1 错误：无效指令  source/file.s:8: Error: bad  instruction `sdd r0, r1’  这个错误发生在你使用个的命令不存在的情况下。首先，你应该检查一下你输入的命令是否有误。如果你使用了条件指令代码，譬如str指令使用了b后缀，以便只存储一个字节，或者使用eq后缀，在最后的条件是相等的情况下存储，正确的指令序列是streqb，而不是strbeq。  1.2 错误：立即表达式需要一个#前缀  source/file.s:32: Error: immediate expression requires a # prefix -- `add r0,1'  这意味着你正在试着使用一个常数，例如把一个数加1，但是忘记在这个常数前加上#符号了（例如：add r3, 4指令的正确版本是add r3, #4。你应该在你使用常数的任何地方都改为使用寄存器。甚至于当计算常数#3 \* 4时也应该这样。）  1.3 错误：预期ARM寄存器  source/file.s:24: Error: ARM register expected -- `add 0,r1'  这意味着你输入的指令里有并不是寄存器的代码，而此时是需要寄存器的。检查你的代码的拼写至少两遍，尤其是使用.req指令时。如果的确是这样，确保你在.req和当前指令之间并没有使用指令.unreq。  1.4 错误：未知的伪操作  source/file.s: Error: unknown pseudo-op: `.suction'  这个操作放生在当你使用一个并不存在的伪操作（伪代码）。这时应该检查你代码的拼写。  1.5 错误：修复之后的无效常数（数值）  source/file.s:24: Error: invalid constant (c21) after fixup  这个错误发生在当你使用一个并不符合函数需求的常数时。这种错误最普通的例子就是mov指令。该指令仅仅允许使用用8位表示的数值。该数值左移一个偶数位。例如c2116 = 1100001000012就无法在指令mov中进行表达，但是c2016 = 1100001000002 = 110000102 << 4就可以在指令mov中使用。还有许多这样的规则应用在函数中大多数的常数中。记住，加载任何常数，请使用指令ldr r0, =value。  1.6 警告：文件的结尾并不是文件内容的最后一行；新行插入  source/file.s: warning : end of file not at end of a line; newline inserted  这意味着你文件的最后一行不是空行。你可以忽略这个警告。如果要修复这个，可以在文件的末尾处增加一个新行。  1.7 |