|  |  |  |
| --- | --- | --- |
| 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. |  |