T Feb 2 - Raspberry Pi Intro

Tuesday, February 2, 2016 11:01 AM

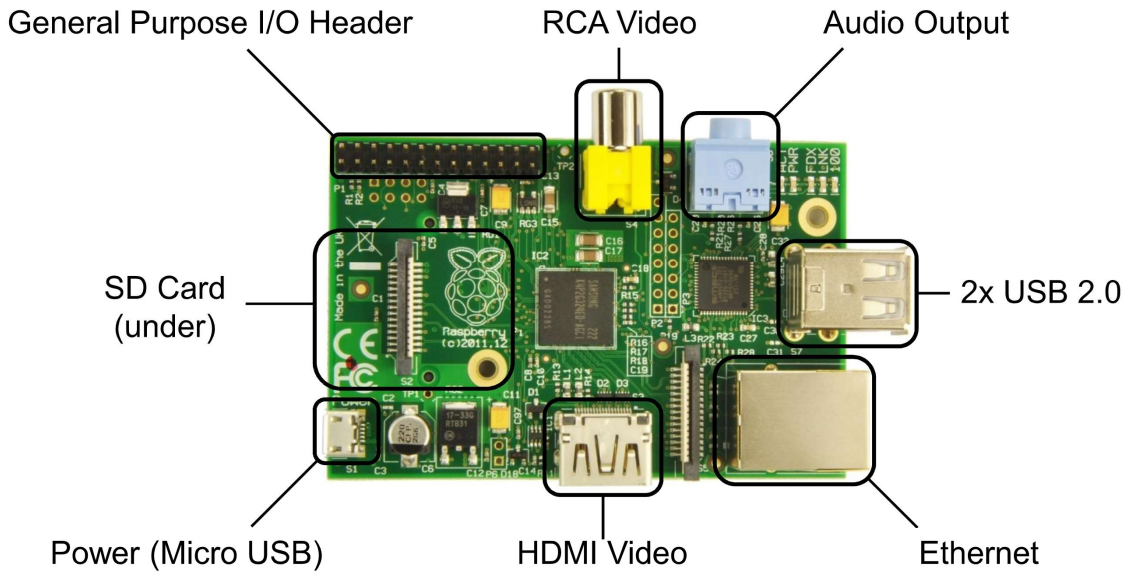
CPSC 359 – Tutorial #1

Introduction to the Raspberry Pi

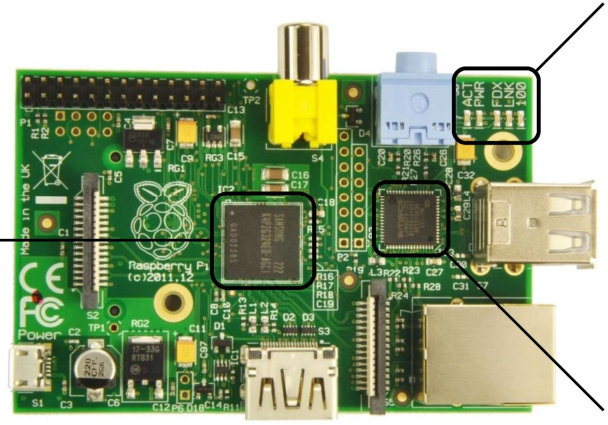
Slide modified from Andrew Kuipers

Last Modified Sept, 2015

Anatomy of the Raspberry Pi



Anatomy of the Raspberry Pi



Status LEDs
ACT – SDCard Access
PWR – Power
FDX – (LAN) Full Duplex
LNK – (LAN) Link
100 – (LAN) 10/100Mb

Broadcom BCM2835
700MHz ARM11 CPU
VideoCore IV GPU

LAN9512
2x USB 2.0
10/100Mb Ethernet

We will be doing bare metal programming on our raspberry pi's.

What is *bare metal* programming?

You *can* put an operating system on the pi. But we wont.

- Writing code to run directly on the hardware
- No Operating System!

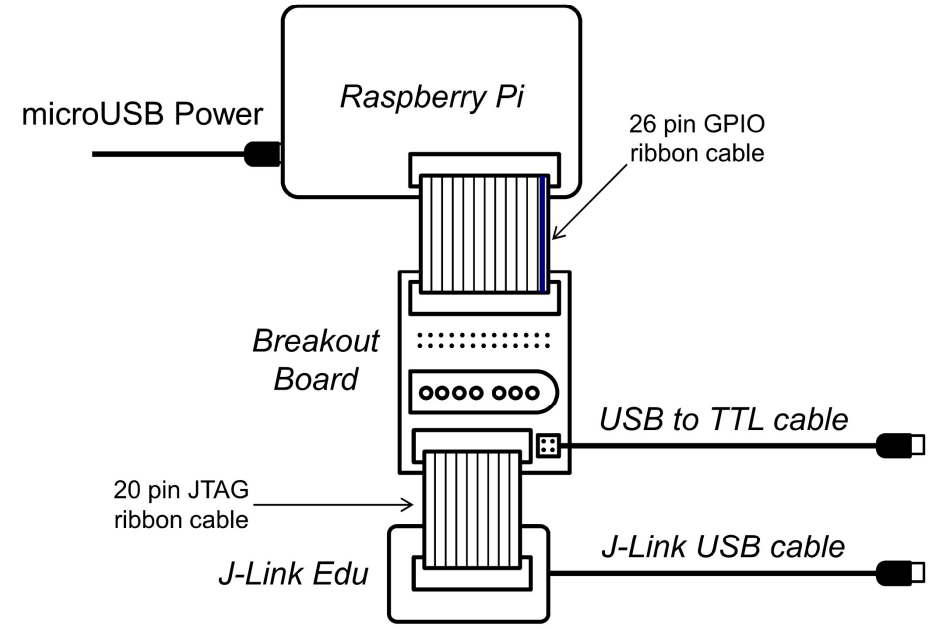
Why write bare metal code?

- Operating Systems prevent direct access to hardware
- Understand fundamentals of the hardware / software interface

What are the drawbacks?

- No standard libraries, need to write everything from scratch
- No protection from OS, could write code that harms the device

JTAG Debugging - Hardware

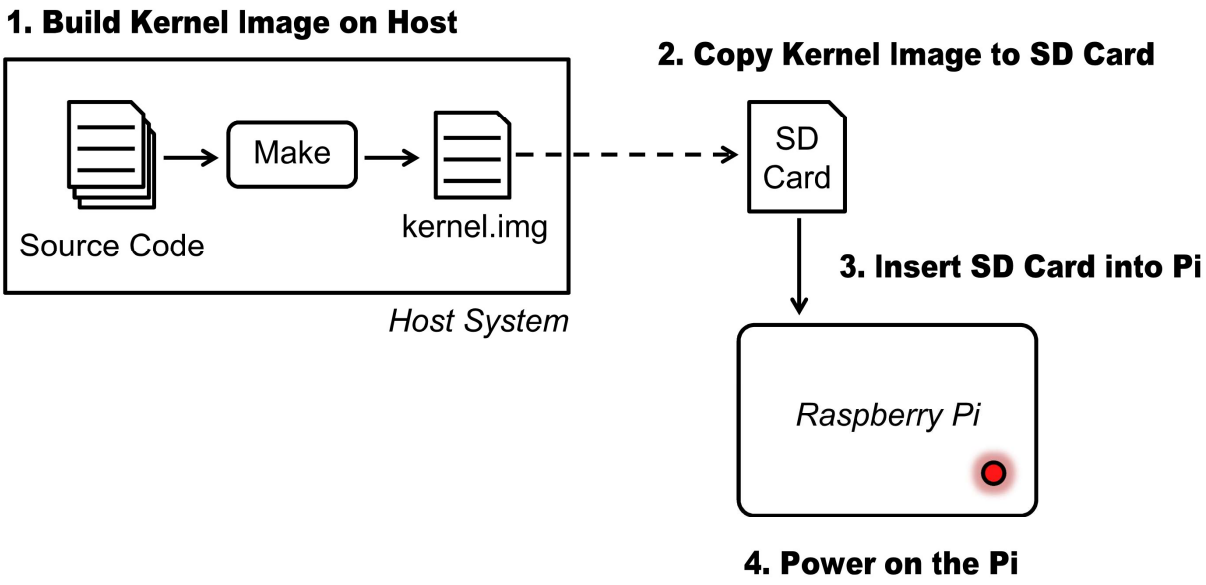


First Time Setup

FYI, it is not needed anymore to be done in the lab…

The following is not necessary anymore. But just interesting to know

Build Procedure – First Time Setup



UNIVERSITY OF CALGARY

Build Procedure – First Time Setup

1. Download and Un-pack Template Project

```
$ mkdir ~/c359
$ cd ~/c359
$ cp /home/courses/359/code/template.tar.gz .
$ tar zxvf template.tar.gz
```

2. Build Kernel Image on Host

DON'T NEED TO DO ANY OF THIS ANYMORE

```
$ cd ~/c359/template/
$ make all
```

3. Format the Card and initialize the partitions (follow the instructions).

```
$ setup_sdcard
```

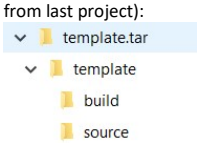
4. Copy Kernel Image to SD Card and Un-mount SD Card

```
$ cp kernel.img /run/media/username/boot/
- Un-mount the SD Card
```

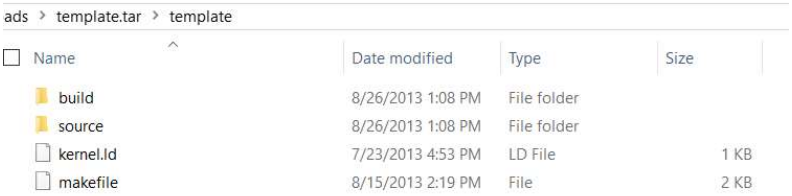
Connection Setup

Needed whenever you connect the RPi…

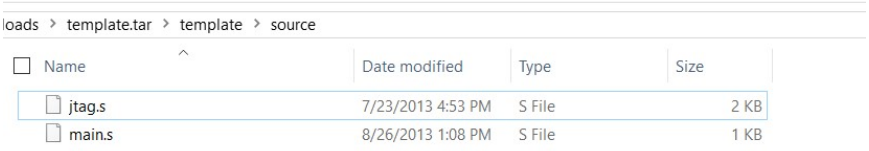
First of all, whenever you want to make a new program, download the following form d2l (or copy it from last project):



It doesn't have to be called template. Just call it the A1 or whatever.
It will be a tar file when downloaded from d2l.
Extract it.
You get a folder called template with 2 folders and 2 files inside.



Build is empty to begin with. That is where object code will go once program is compiled.
Kernel.ld and makefile will be the exact same for every project. Don't touch them.
Inside source..



Inside source there are 2 files.
Jtag.s is the same for every project and is written for us already. Don't touch it.
Main.s is the assemply code file. Here is where you will write your program.

```
.section    .init
.globl     _start

_start:
    b       main

.section .text

main:
    mov     sp, #0x8000

    bl      EnableJTAG

    ////////////////////
    //WRITE CODE HERE
    ////////////////////

haltLoop$:
    b       haltLoop$

.section .data

```

^main.s. write code where the comment says to do so. Don't erase any other things. They are important.

Now see how to compile and run whatever you code in main.s:

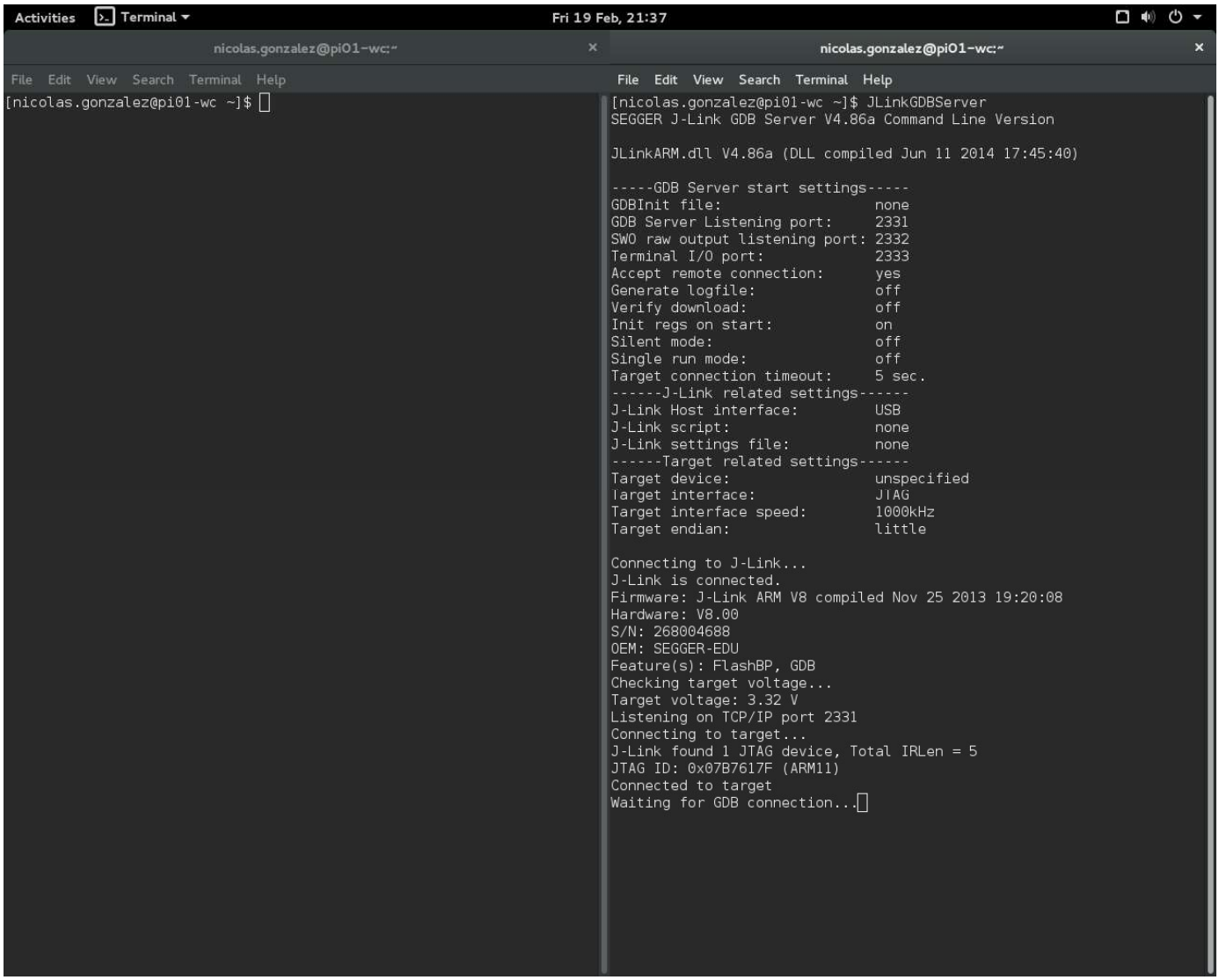
1. Connect 26pin Ribbon Cable from Pi to Breakout Board

2. Connect microUSB Power Cable to the Pi

3. Start J-Link GDB Server in its own terminal window

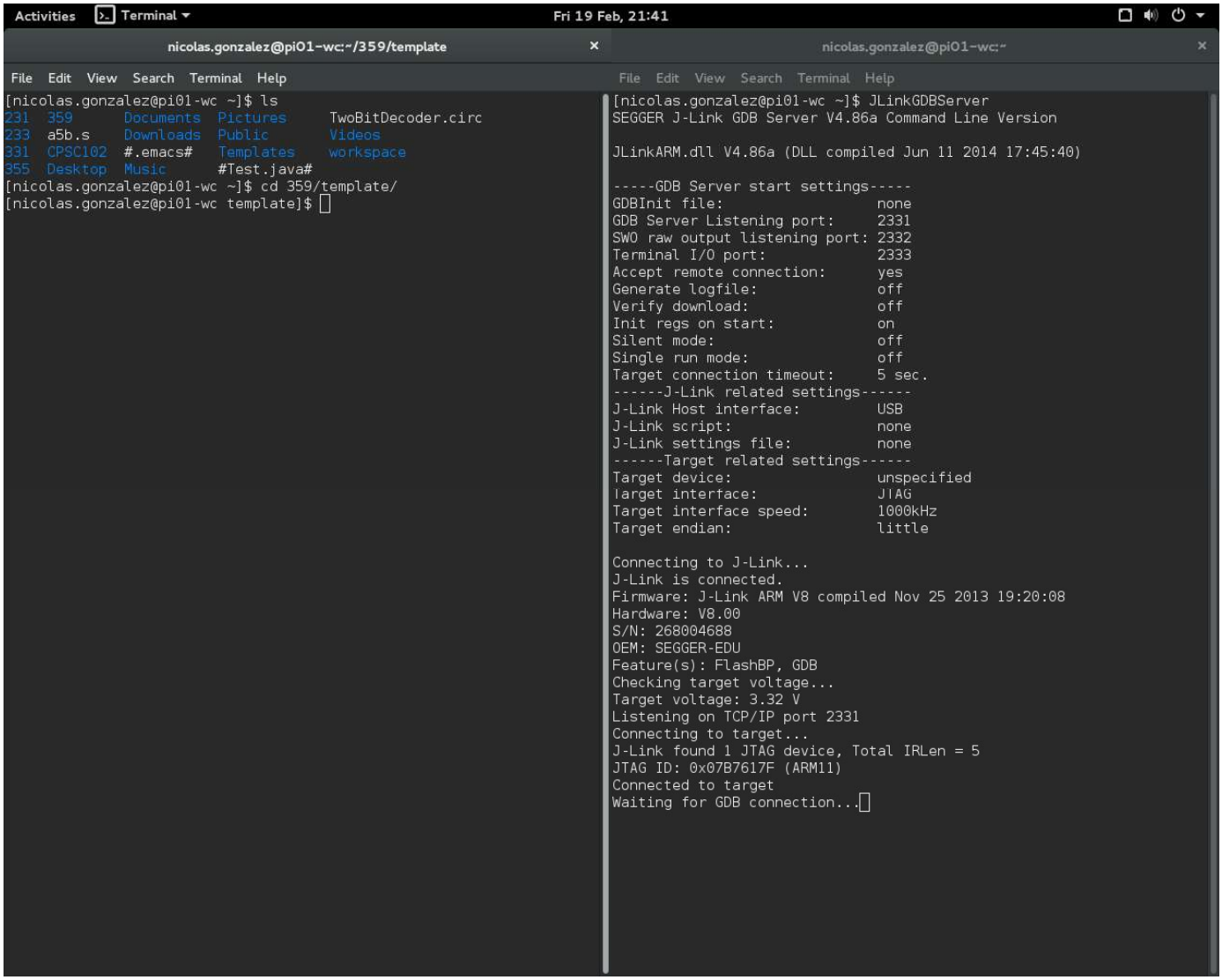
```
$ JLinkGDBServer
```

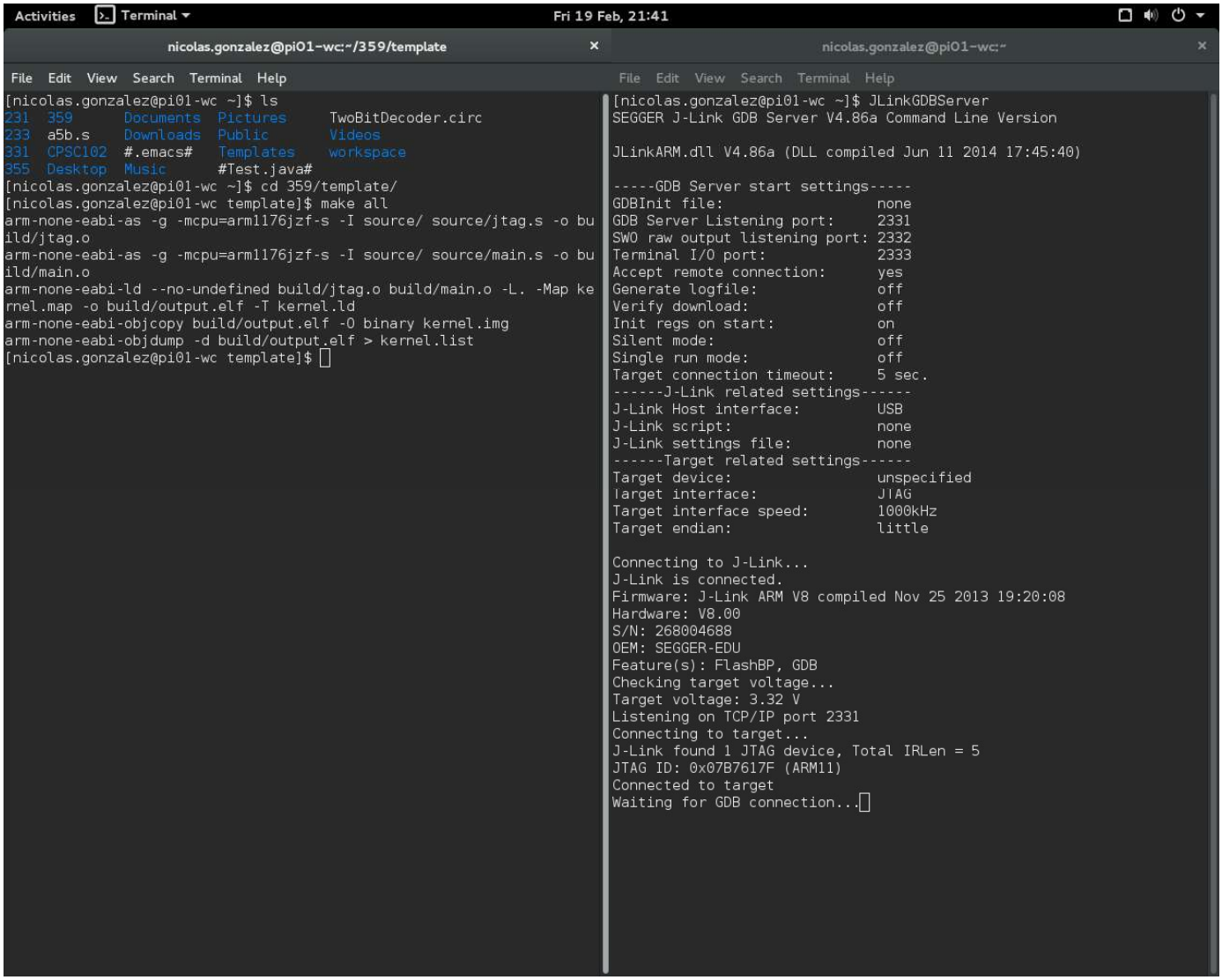
Should get some stuff and at the end:
Connected to target
Waiting for GDB connection...

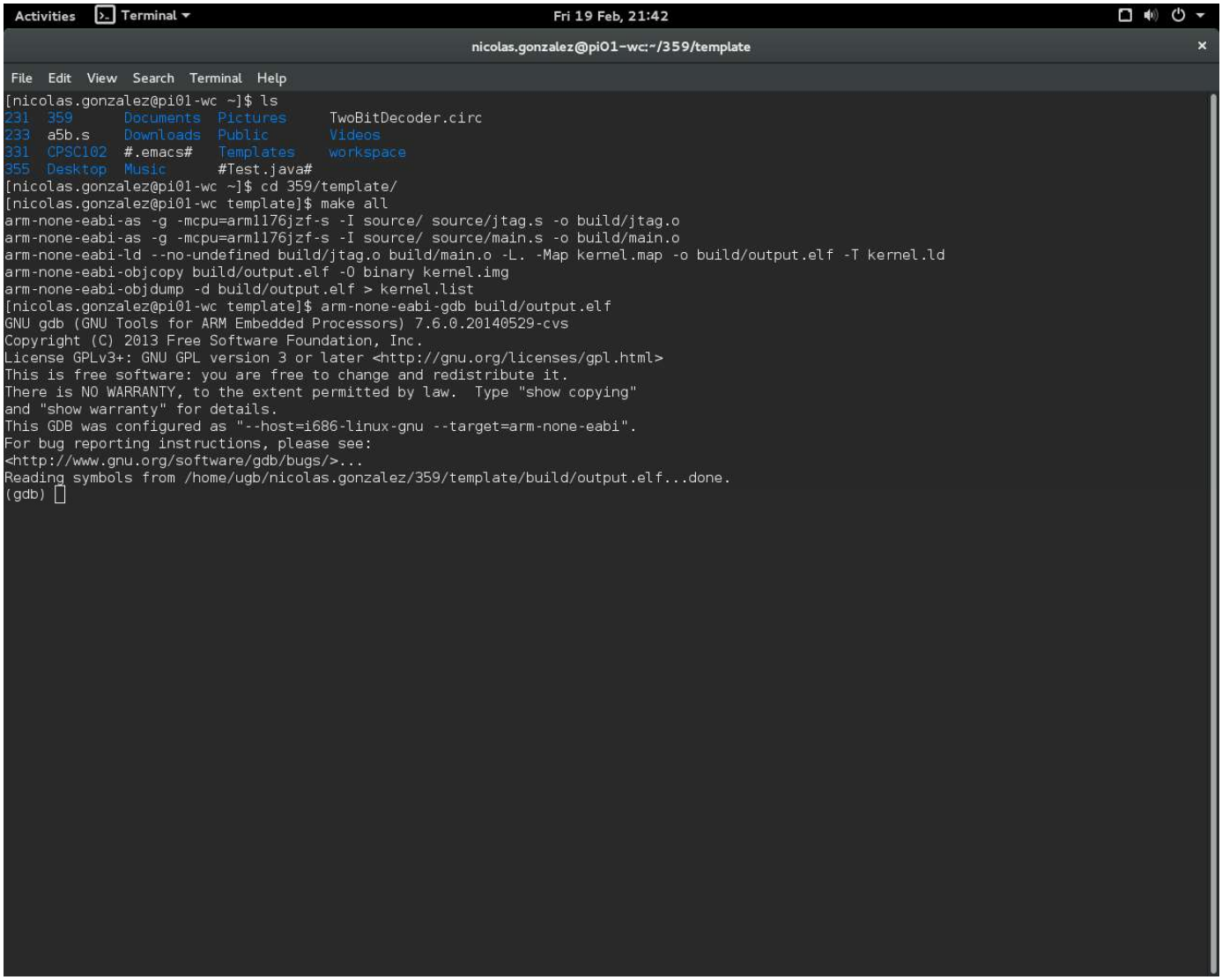


4. Start GDB in the same folder as your *makefile* (after building project) Another terminal window

```
$ cd ~/c359/template/
$ make all
$ arm-none-eabi-gdb build/output.elf
```
Start gdb

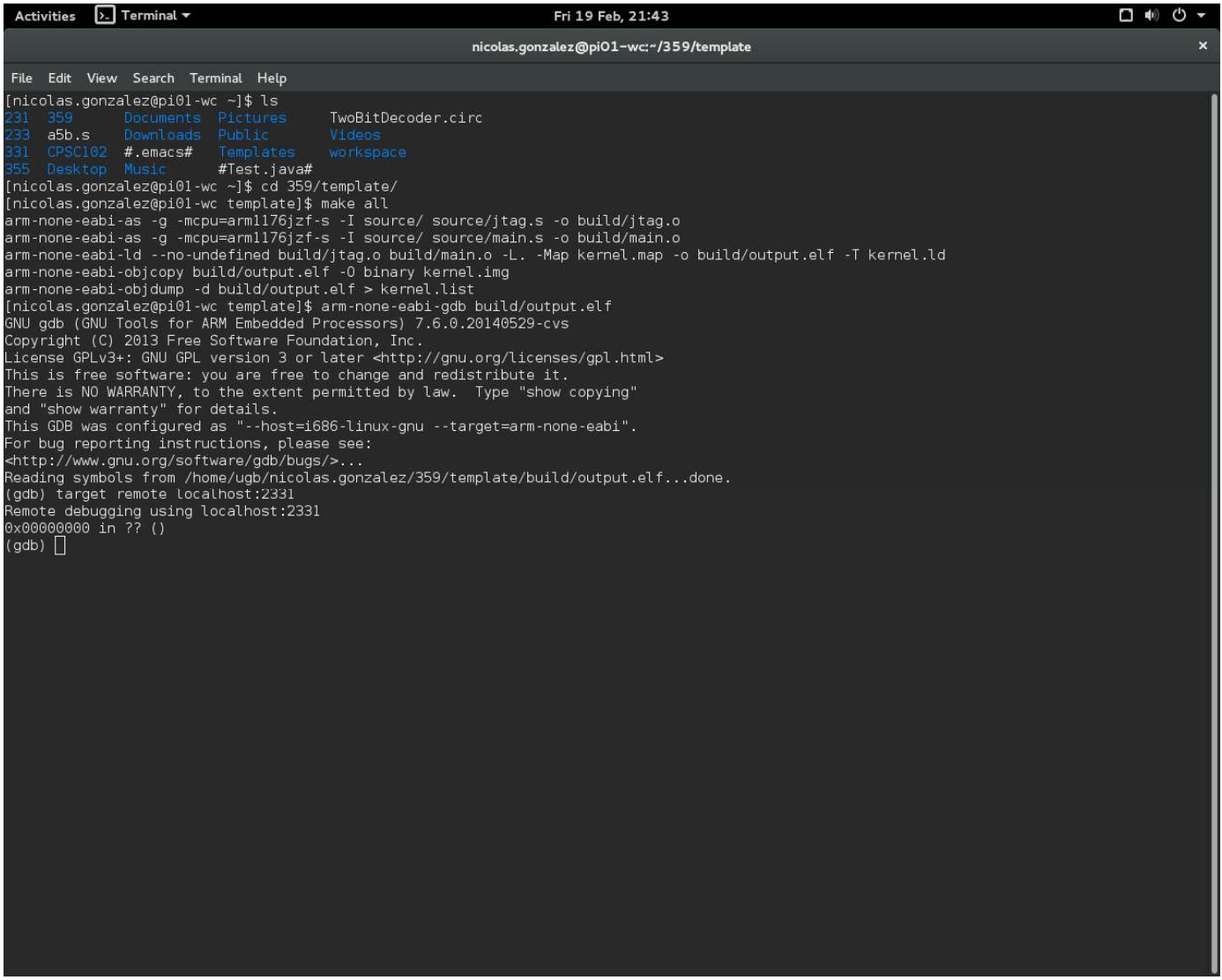

Activities
Terminal
Fri 19 Feb, 21:41
nicolas.gonzalez@pi01-wc:~/359/template
File Edit View Search Terminal Help
[nicolas.gonzalez@pi01-wc ~]$ ls
231 359 Documents Pictures TwoBitDecoder.circ
233 a5b.s Downloads Public Videos
331 CPSC102 #.emacs# Templates workspace
355 Desktop Music #Test.java#
[nicolas.gonzalez@pi01-wc ~]$ cd 359/template/
[nicolas.gonzalez@pi01-wc template]$
nicolas.gonzalez@pi01-wc:~
File Edit View Search Terminal Help
[nicolas.gonzalez@pi01-wc ~]$ JLinkGDBServer
SEGGER J-Link GDB Server V4.86a Command Line Version
JLinkARM.dll V4.86a (DLL compiled Jun 11 2014 17:45:40)
-----GDB Server start settings-----
GDBInit file: none
GDB Server Listening port: 2331
SWO raw output listening port: 2332
Terminal I/O port: 2333
Accept remote connection: yes
Generate logfile: off
Verify download: off
Init regs on start: on
Silent mode: off
Single run mode: off
Target connection timeout: 5 sec.
------J-Link related settings------
J-Link Host interface: USB
J-Link script: none
J-Link settings file: none
------Target related settings------
Target device: unspecified
Target interface: JTAG
Target interface speed: 1000kHz
Target endian: little
Connecting to J-Link...
J-Link is connected.
Firmware: J-Link ARM V8 compiled Nov 25 2013 19:20:08
Hardware: V8.00
S/N: 268004688
OEM: SEGGER-EDU
Feature(s): FlashBP, GDB
Checking target voltage...
Target voltage: 3.32 V
Listening on TCP/IP port 2331
Connecting to target...
J-Link found 1 JTAG device, Total IRLen = 5
JTAG ID: 0x07B7617F (ARM11)
Connected to target
Waiting for GDB connection...


Activities
Terminal
Fri 19 Feb, 21:41
nicolas.gonzalez@pi01-wc:~/359/template
File Edit View Search Terminal Help
[nicolas.gonzalez@pi01-wc ~]$ ls
231 359 Documents Pictures TwoBitDecoder.circ
233 a5b.s Downloads Public Videos
331 CPSC102 #.emacs# Templates workspace
355 Desktop Music #Test.java#
[nicolas.gonzalez@pi01-wc ~]$ cd 359/template/
[nicolas.gonzalez@pi01-wc template]$ make all
arm-none-eabi-as -g -mcpu=arm1176jzf-s -I source/ source/jtag.s -o build/jtag.o
arm-none-eabi-as -g -mcpu=arm1176jzf-s -I source/ source/main.s -o build/main.o
arm-none-eabi-ld --no-undefined build/jtag.o build/main.o -L. -Map kernel.map -o build/output.elf -T kernel.ld
arm-none-eabi-objcopy build/output.elf -O binary kernel.img
arm-none-eabi-objdump -d build/output.elf > kernel.list
[nicolas.gonzalez@pi01-wc template]$
nicolas.gonzalez@pi01-wc:~
File Edit View Search Terminal Help
[nicolas.gonzalez@pi01-wc ~]$ JLinkGDBServer
SEGGER J-Link GDB Server V4.86a Command Line Version
JLinkARM.dll V4.86a (DLL compiled Jun 11 2014 17:45:40)
-----GDB Server start settings-----
GDBInit file: none
GDB Server Listening port: 2331
SWO raw output listening port: 2332
Terminal I/O port: 2333
Accept remote connection: yes
Generate logfile: off
Verify download: off
Init regs on start: on
Silent mode: off
Single run mode: off
Target connection timeout: 5 sec.
------J-Link related settings------
J-Link Host interface: USB
J-Link script: none
J-Link settings file: none
------Target related settings------
Target device: unspecified
Target interface: JTAG
Target interface speed: 1000kHz
Target endian: little
Connecting to J-Link...
J-Link is connected.
Firmware: J-Link ARM V8 compiled Nov 25 2013 19:20:08
Hardware: V8.00
S/N: 268004688
OEM: SEGGER-EDU
Feature(s): FlashBP, GDB
Checking target voltage...
Target voltage: 3.32 V
Listening on TCP/IP port 2331
Connecting to target...
J-Link found 1 JTAG device, Total IRLen = 5
JTAG ID: 0x07B7617F (ARM11)
Connected to target
Waiting for GDB connection...



5. Connect GDB to J-Link GDB Server

```
(gdb) target remote localhost:2331
```


Activities
Terminal
Fri 19 Feb, 21:43
nicolas.gonzalez@pi01-wc:~/359/template
File Edit View Search Terminal Help
[nicolas.gonzalez@pi01-wc ~]$ ls
231 359 Documents Pictures TwoBitDecoder.circ
233 a5b.s Downloads Public Videos
331 CPSC102 #.emacs# Templates workspace
355 Desktop Music #Test.java#
[nicolas.gonzalez@pi01-wc ~]$ cd 359/template/
[nicolas.gonzalez@pi01-wc template]$ make all
arm-none-eabi-as -g -mcpu=arm1176jzf-s -I source/ source/jtag.s -o build/jtag.o
arm-none-eabi-as -g -mcpu=arm1176jzf-s -I source/ source/main.s -o build/main.o
arm-none-eabi-ld --no-undefined build/jtag.o build/main.o -L. -Map kernel.map -o build/output.elf -T kernel.ld
arm-none-eabi-objcopy build/output.elf -O binary kernel.img
arm-none-eabi-objdump -d build/output.elf > kernel.list
[nicolas.gonzalez@pi01-wc template]$ arm-none-eabi-gdb build/output.elf
GNU gdb (GNU Tools for ARM Embedded Processors) 7.6.0.20140529-cvs
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-linux-gnu --target=arm-none-eabi".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/ugb/nicolas.gonzalez/359/template/build/output.elf...done.
(gdb) target remote localhost:2331
Remote debugging using localhost:2331
0x00000000 in ?? ()
(gdb)

Compiling

Needed whenever you want to compile your code…

JTAG Debugging – Remote GDB

Open *source/main.s* in a text editor and modify:

```
    bl      EnableJTAG
    mov     r0, #42             // <-- add this line
haltLoop$:
```

This is just a random line of code.
If youre following the black pictures. I actually did mov r1, #5

Back in gdb:

1. Build the project: `(gdb) make`
2. Upload new image to Pi: `(gdb) load`
3. Set program counter to beginning: `(gdb) set $pc = _start`
4. Continue execution: `(gdb) continue`

These last 2 commands (3 and 4) can be replaced by:
(gdb)j _start

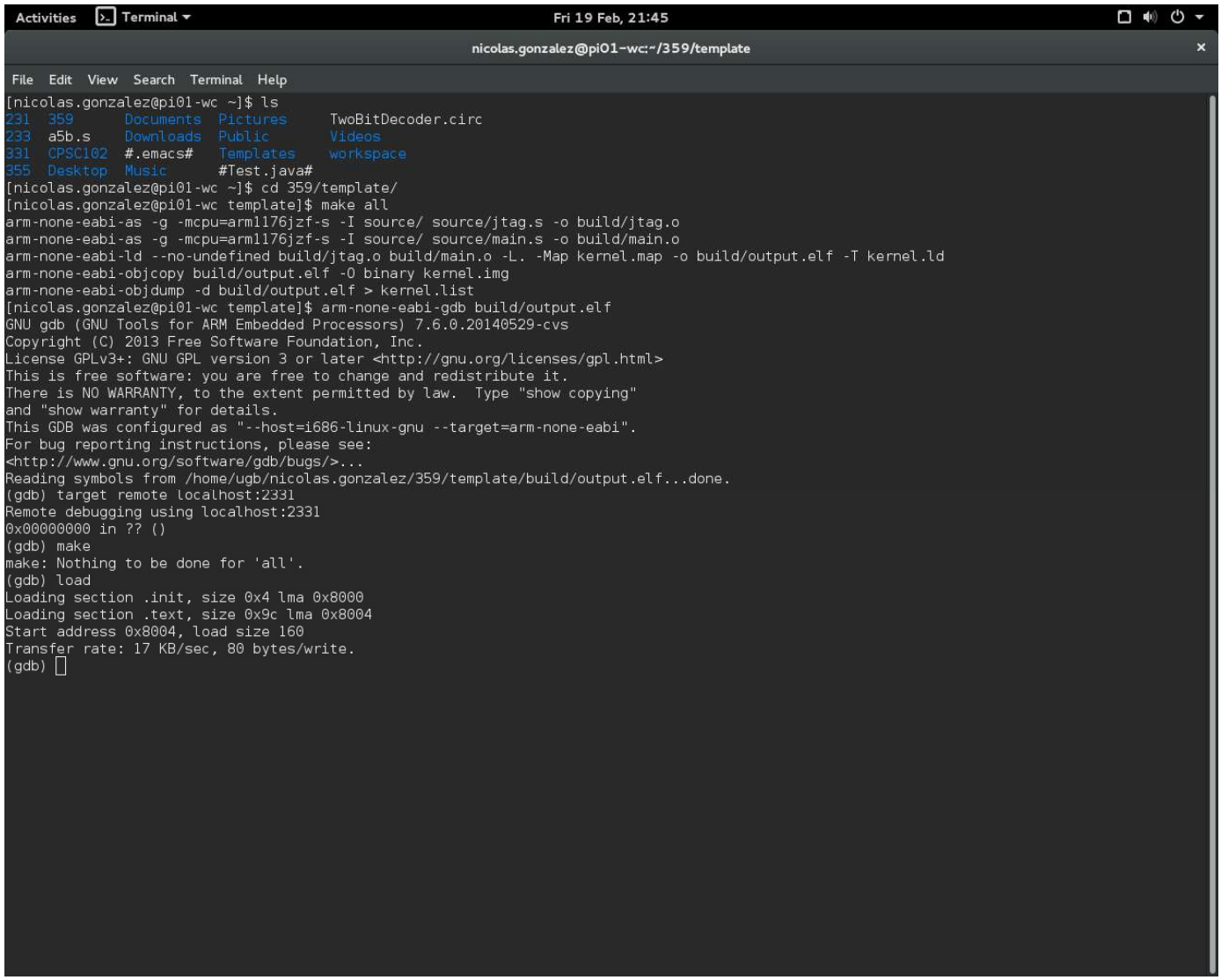
Ctrl+C to exit program //get it out of infinite halt loop

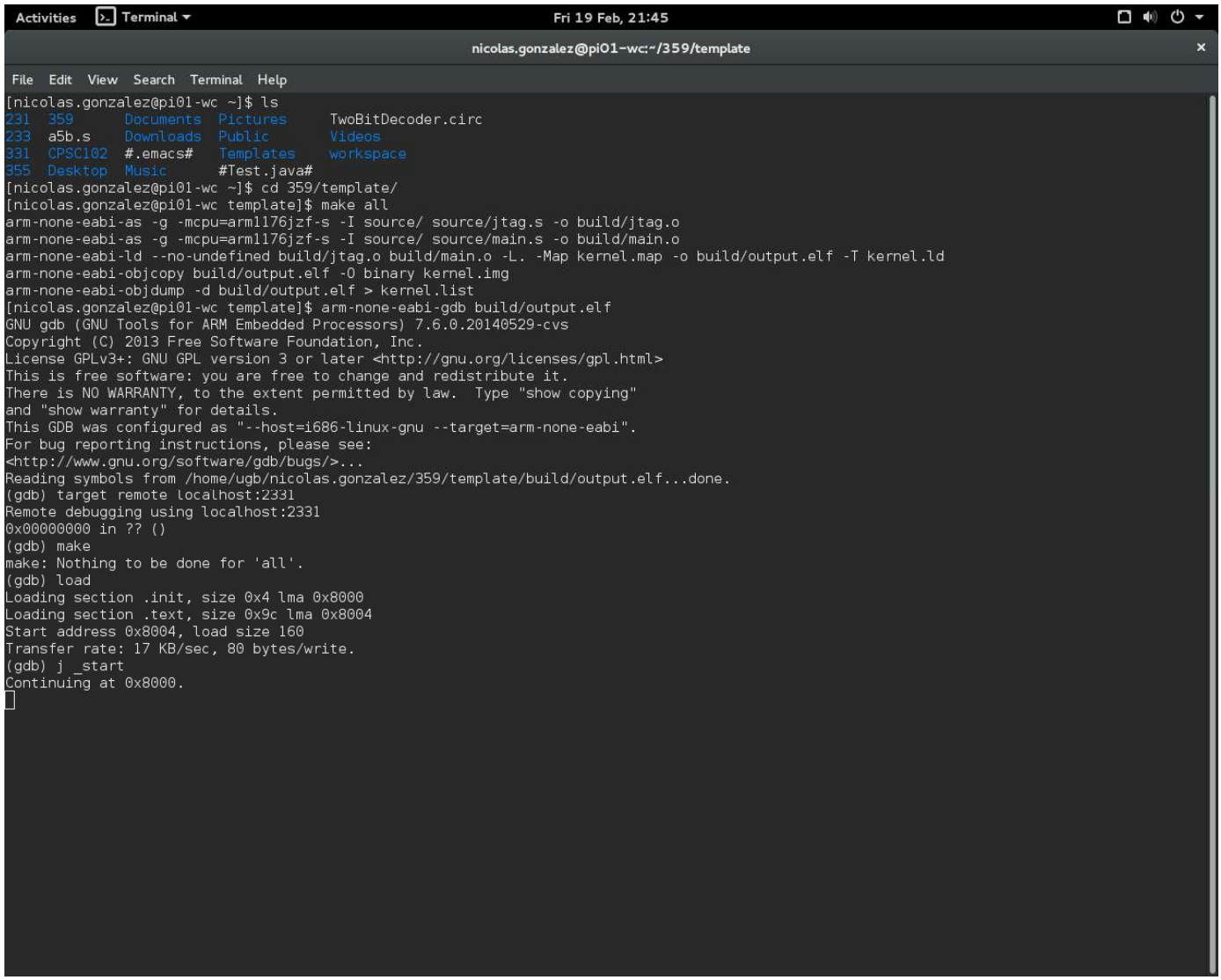
(gdb) info registers //to get register contents

To exit:
(gdb) disconnect
(gdb) quit

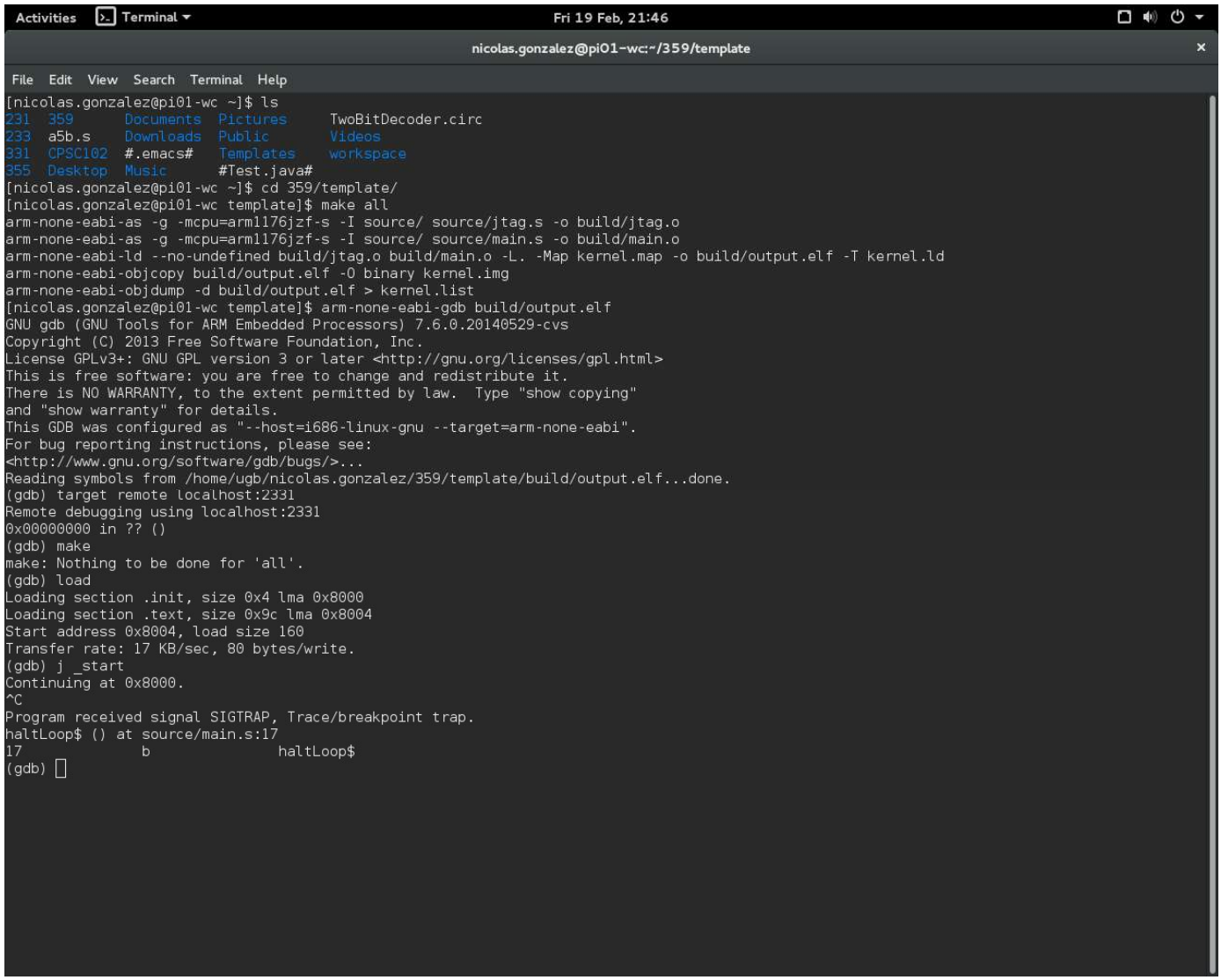
Unplug pi first.
Unplug jtag second

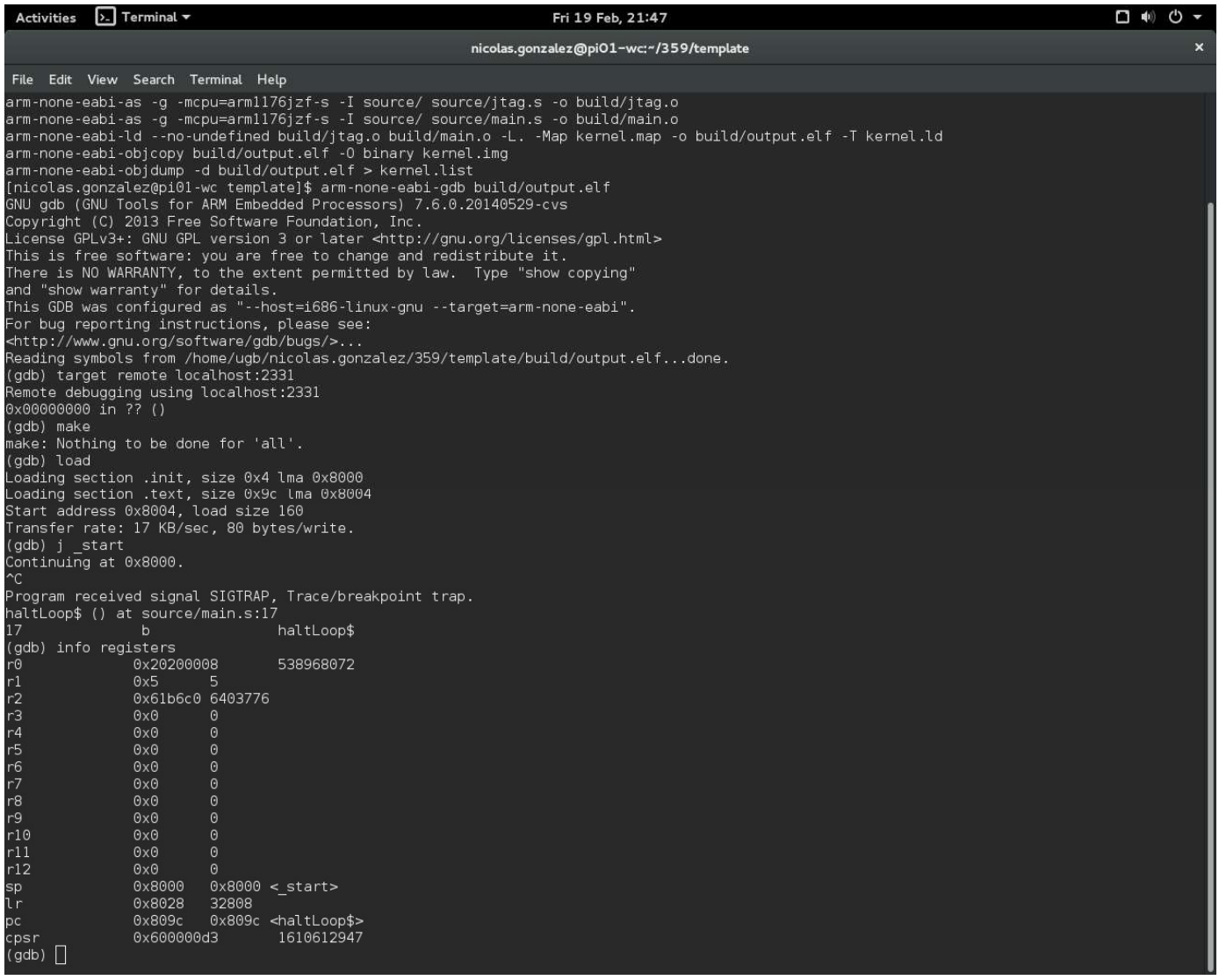
To set breakpoints:
After doing
(gdb) load
You can set breakpoints by:
(gdb) b <label>
Then to continue to the next break point do
(gdb) continue
When at breakpoint, you can examine memory, print registers, do "info registers" etc.
This is all done for debugging!


Activities
Terminal
Fri 19 Feb, 21:45
nicolas.gonzalez@pi01-wc:~/359/template
File Edit View Search Terminal Help
[nicolas.gonzalez@pi01-wc ~]$ ls
231 359 Documents Pictures TwoBitDecoder.circ
233 a5b.s Downloads Public Videos
331 CPSC102 #.emacs# Templates workspace
355 Desktop Music #Test.java#
[nicolas.gonzalez@pi01-wc ~]$ cd 359/template/
[nicolas.gonzalez@pi01-wc template]$ make all
arm-none-eabi-as -g -mcpu=arm1176jzf-s -I source/ source/jtag.s -o build/jtag.o
arm-none-eabi-as -g -mcpu=arm1176jzf-s -I source/ source/main.s -o build/main.o
arm-none-eabi-ld --no-undefined build/jtag.o build/main.o -L. -Map kernel.map -o build/output.elf -T kernel.ld
arm-none-eabi-objcopy build/output.elf -O binary kernel.img
arm-none-eabi-objdump -d build/output.elf > kernel.list
[nicolas.gonzalez@pi01-wc template]$ arm-none-eabi-gdb build/output.elf
GNU gdb (GNU Tools for ARM Embedded Processors) 7.6.0.20140529-cvs
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-linux-gnu --target=arm-none-eabi".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/ugb/nicolas.gonzalez/359/template/build/output.elf...done.
(gdb) target remote localhost:2331
Remote debugging using localhost:2331
0x00000000 in ?? ()
(gdb) make
make: Nothing to be done for 'all'.
(gdb) load
Loading section .init, size 0x4 lma 0x8000
Loading section .text, size 0x9c lma 0x8004
Start address 0x8004, load size 160
Transfer rate: 17 KB/sec, 80 bytes/write.
(gdb)

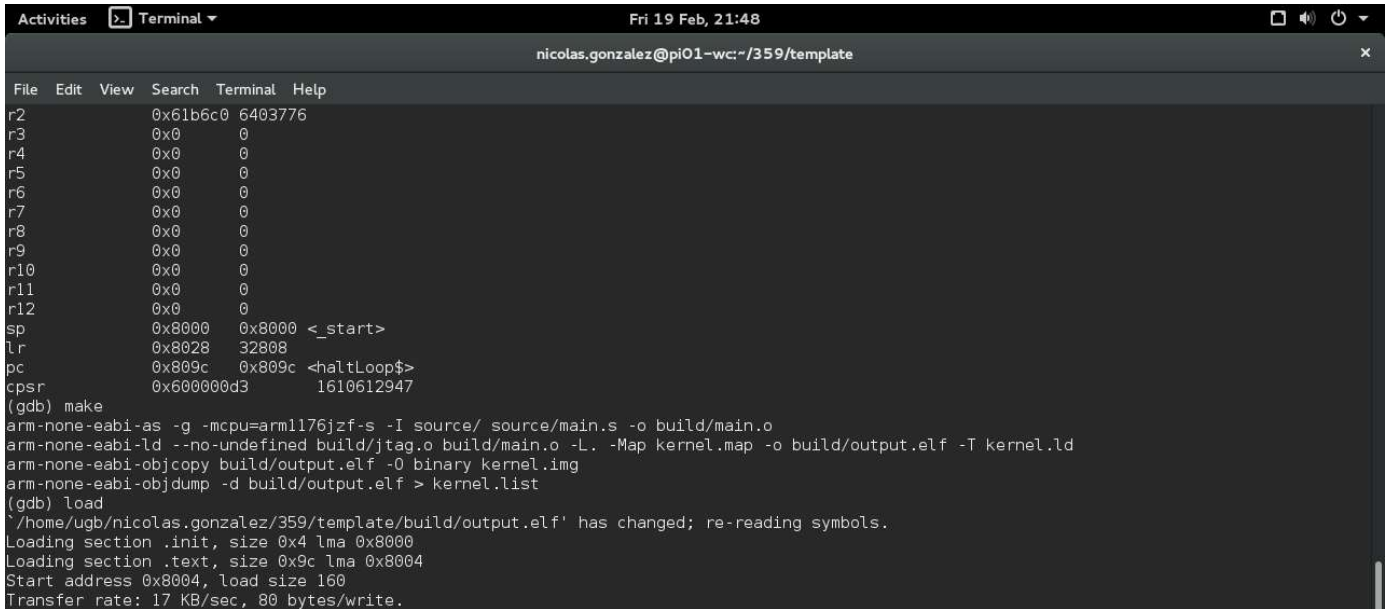


^in infinite loop. Now do ctrl+C to exit infinite loop:


Activities
Terminal
Fri 19 Feb, 21:46
nicolas.gonzalez@pi01-wc:~/359/template
File Edit View Search Terminal Help
[nicolas.gonzalez@pi01-wc ~]$ ls
231 359 Documents Pictures TwoBitDecoder.circ
233 a5b.s Downloads Public Videos
331 CPSC102 #.emacs# Templates workspace
355 Desktop Music #Test.java#
[nicolas.gonzalez@pi01-wc ~]$ cd 359/template/
[nicolas.gonzalez@pi01-wc template]$ make all
arm-none-eabi-as -g -mcpu=arm1176jzf-s -I source/ source/jtag.s -o build/jtag.o
arm-none-eabi-as -g -mcpu=arm1176jzf-s -I source/ source/main.s -o build/main.o
arm-none-eabi-ld --no-undefined build/jtag.o build/main.o -L. -Map kernel.map -o build/output.elf -T kernel.ld
arm-none-eabi-objcopy build/output.elf -O binary kernel.img
arm-none-eabi-objdump -d build/output.elf > kernel.list
[nicolas.gonzalez@pi01-wc template]$ arm-none-eabi-gdb build/output.elf
GNU gdb (GNU Tools for ARM Embedded Processors) 7.6.0.20140529-cvs
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-linux-gnu --target=arm-none-eabi".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/ugb/nicolas.gonzalez/359/template/build/output.elf...done.
(gdb) target remote localhost:2331
Remote debugging using localhost:2331
0x00000000 in ?? ()
(gdb) make
make: Nothing to be done for 'all'.
(gdb) load
Loading section .init, size 0x4 lma 0x8000
Loading section .text, size 0x9c lma 0x8004
Start address 0x8004, load size 160
Transfer rate: 17 KB/sec, 80 bytes/write.
(gdb) j _start
Continuing at 0x8000.
^C
Program received signal SIGTRAP, Trace/breakpoint trap.
haltLoop$ () at source/main.s:17
17 b haltLoop$
(gdb)

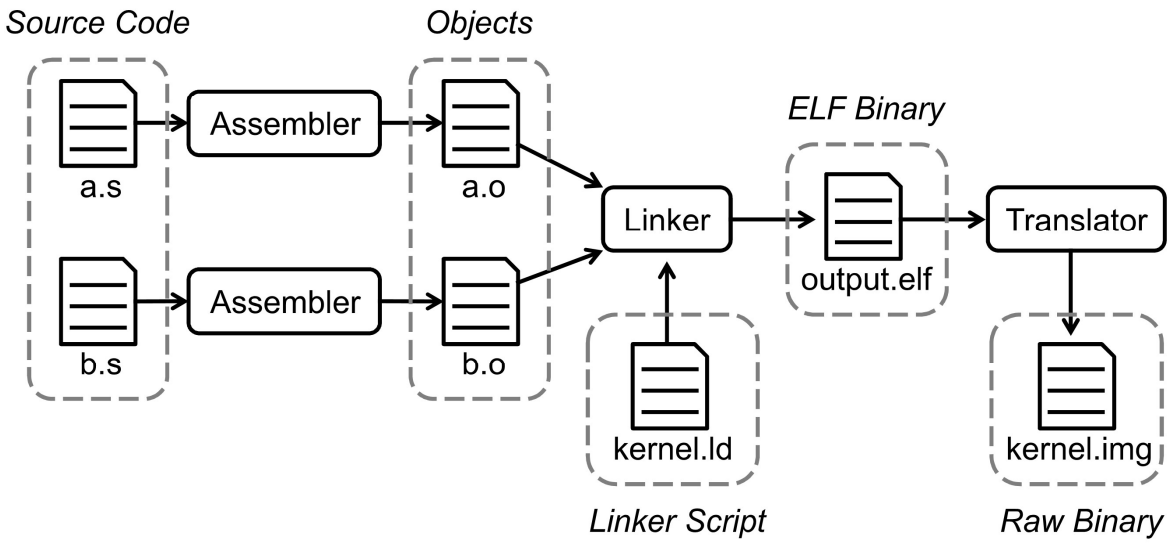


Notice 5 in in r1. just as we expected.
Now I went back to the source code and changed that line to mov r0, #42.
Now we can recompile without leaving gdb. Just call make again:
Then load
Then start it
Then ctrl C.
Then examine registers



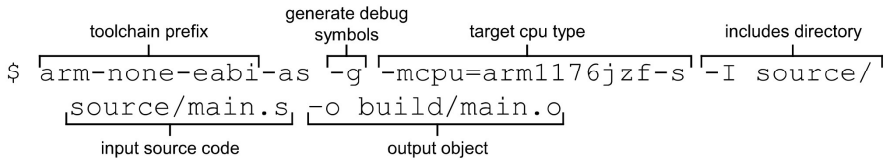
```
`/home/ugb/nicolas.gonzalez/359/template/build/output.elf' has changed; re-reading symbols.
Loading section .init, size 0x4 lma 0x8000
Loading section .text, size 0x9c lma 0x8004
Start address 0x8004, load size 160
Transfer rate: 17 KB/sec, 80 bytes/write.
(gdb) j _start
Continuing at 0x8000.
^C
Program received signal SIGTRAP, Trace/breakpoint trap.
haltLoop$ () at source/main.s:17
17              b               haltLoop$
(gdb) info registers
r0             0x2a     42
r1             0x61b6c8 6403784
r2             0x61b6c0 6403776
r3             0x0      0
r4             0x0      0
r5             0x0      0
r6             0x0      0
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0x0      0
r11            0x0      0
r12            0x0      0
sp             0x8000   0x8000 <_start>
lr             0x8028   32808
pc             0x809c   0x809c <haltLoop$>
cpsr           0x600000d3       1610612947
(gdb)
```

What is the Makefile doing?

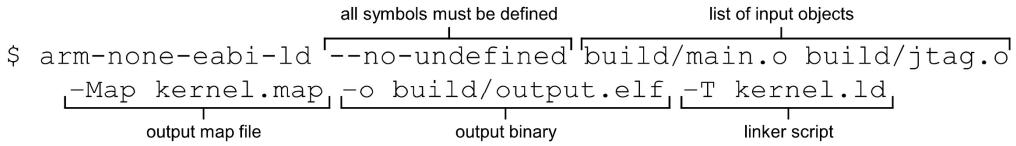


What is the Makefile doing?

1. Assemble Source Code to Objects



2. Link Objects into ELF Binary



3. Translate ELF Binary to Raw Binary