Assignment I Report

Cross-Platform and Toolchain Customization

20th March, 2025

1. MIPS and ARM architecture differences

Architecturel Diffrenceses :

Feature	MIPS	ARM
Instruction Set	Fixed 32-bit instructions	32-bit and 16-bit (Thumb mode)
Registers	32 general-purpose registers	16 (32-bit mode), 31 (64-bit mode)
Addressing Modes	Fewer (e.g., base+offset)	More (e.g., auto-increment)
Conditional Execution	Uses delay slots for branching	Supports conditional instructions
Pipeline	Simple 5-stage pipeline	Variable, often more advanced

MIPS Applications :

MIPS is mainly used in embedded systems and networking because of its reduced cost compared to other processors. It is also used in other applications, including video game consoles like Nintendo and Sony Play Station, as it supports graphics and long-term support.

ARM Application :

ARM is used in all technical fields, particularly in consumer electronic devices like mobile devices, tablets and multimedia players. The reason behind this is because of its low power consumption, high performance and low implementation cost.

2. Difference between cross-compiling toolchain and native tool chain.

- A cross-compiling toolchain is used to compile code on one system (host) but run it on another (target). This is essential for embedded systems, where compiling directly on the target is impractical due to limited resources.

- A native toolchain, on the other hand, compiles and runs code on the same system. For example, compiling a C program on a Linux PC using gcc and running it on the same machine is an example of native compilation.

To Sum-up, Cross-compiling ensures that developers can build software for devices with different CPU architectures, such as MIPS or ARM, without needing to run the compiler on the target device itself.

3A. (Bootloader, kernel, and filesystem) Definitions

Bootload :

- A bootloader is a small program that runs before the operating system starts. It initializes the hardware and loads the kernel into memory.
- **U-Boot in Linux and Winboot in windows.**

Kernal :

- The kernel is the core component of an operating system. It manages hardware resources, provides essential system services, and enables communication between hardware and software.

Filesystem :

- A filesystem is a structured way of storing and organizing files on a device.

3B. Usage of Kernel modules.

Kernel modules are pieces of code that can be loaded and unloaded into the kernel upon demand. “They extend the functionality of the kernel without the need to reboot the system”

Examples and usage :

- Add device drivers
- Enable additional filesystems.
- Provide kernel-level features. “Such as Security”
- Avoid rebuilding the entire kernel for updates or new functionality.

4. MIPS Cross-Compiling Toolchain

Step1:

I modified the hello.c application to take a name parameter and greet him/her.

Then I made sure that my apt and is updated.

I did download Crosstool-ng during the lab.

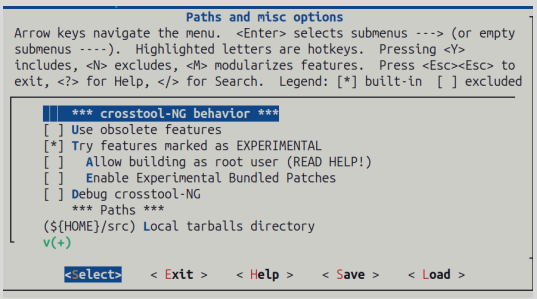
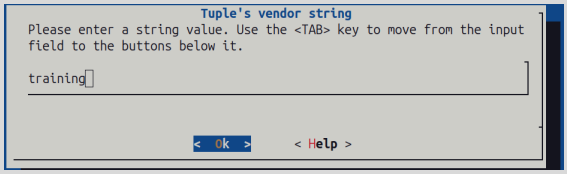
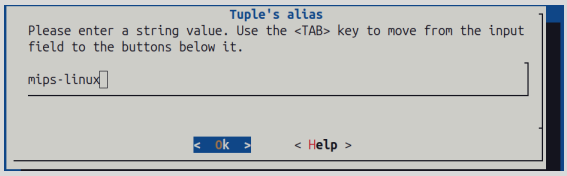
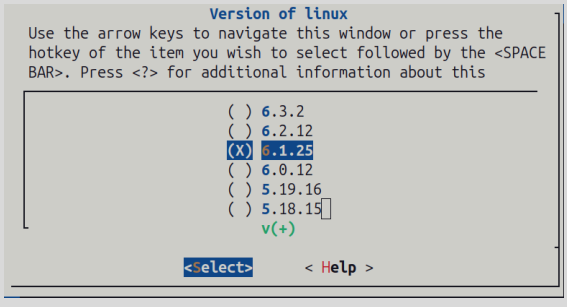
GNU nano 7.2

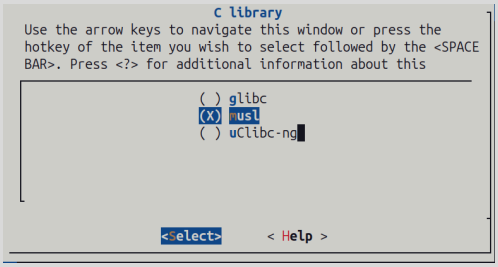
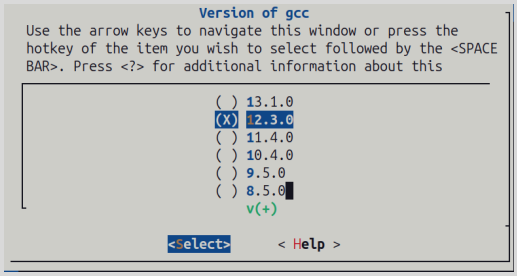
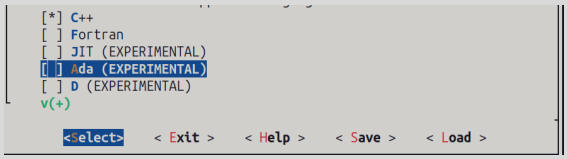
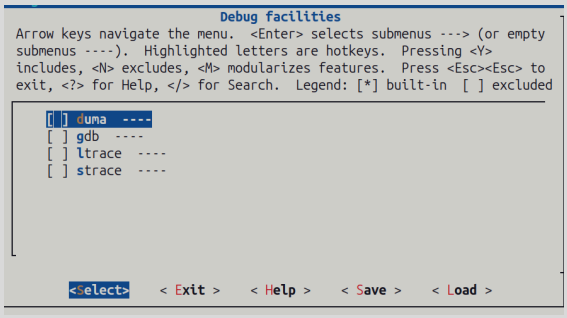
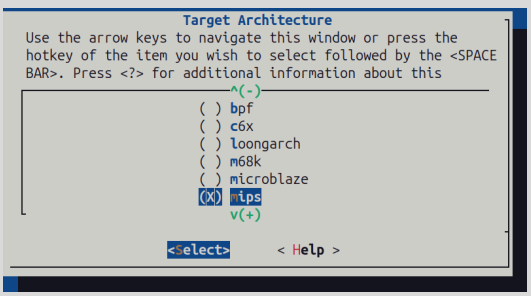
```c
#include <stdio.h>

int main(int argc, char **argv) {
    if (argc != 2) {
        printf("No Parameter! \n");
        return 1;
    }
    printf("Hello %s!\n", argv[1]);
        return EXIT_SUCCESS;
}
```

Next, I need to generate a configure script using ./bootstrap command

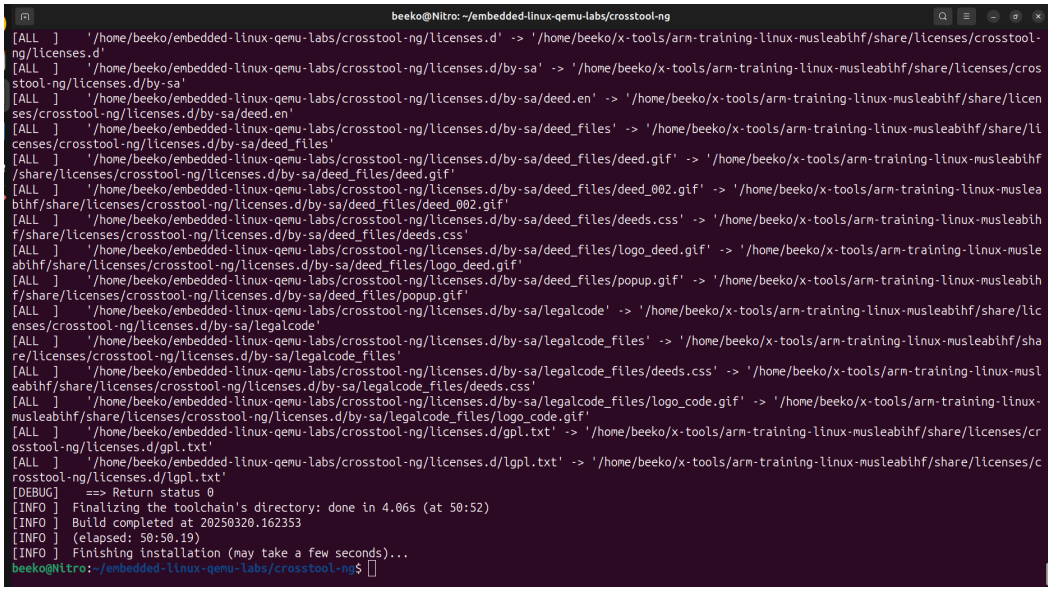
```
beeko@Nitro:~/embedded-linux-qemu-labs/crosstool-ng$ ./bootstrap
INFO  :: *** Generating package version descriptions
INFO  :: Master packages: autoconf automake avr-libc binutils bison cloog dtc duma elf2flt expat gcc gdb gettext glibc gmp gnuprumcu isl libelf libiconv
libtool linux ltrace m4 make mingw-w64 moxiebox mpc mpfr musl ncurses newlib-nano newlib picolibc strace uClibc-ng zlib zstd
INFO  :: Generating 'config/versions/autoconf.in'
INFO  :: Generating 'config/versions/automake.in'
INFO  :: Generating 'config/versions/avr-libc.in'
INFO  :: Generating 'config/versions/binutils.in'
INFO  :: Generating 'config/versions/bison.in'
INFO  :: Generating 'config/versions/cloog.in'
INFO  :: Generating 'config/versions/dtc.in'
INFO  :: Generating 'config/versions/duma.in'
INFO  :: Generating 'config/versions/elf2flt.in'
INFO  :: Generating 'config/versions/expat.in'
INFO  :: Generating 'config/versions/gcc.in'
INFO  :: Generating 'config/versions/gdb.in'
INFO  :: Generating 'config/versions/gettext.in'
INFO  :: Generating 'config/versions/glibc.in'
INFO  :: Generating 'config/versions/gmp.in'
INFO  :: Generating 'config/versions/gnuprumcu.in'
INFO  :: Generating 'config/versions/isl.in'
INFO  :: Generating 'config/versions/libelf.in'
INFO  :: Generating 'config/versions/libiconv.in'
```

After that, I open the menuconfig and edit the configuration as the following:

	Screen Shot
Enable Try features marked as EXPERIMENTAL	
Tuple's vendor string to training.	
Tuple's alias to mips-linux	
OS Version of Linux to the 6.1.x version	

C library to musl	
C- Compiler Version to gcc to 12.3.0	
C++ (CC_LANG_CXX) is enabled	
Remove Debug facilities options	
Change the Target Arch to MIPS	

Now I can build/Produce the toolchain,



After that I generated the mips binary and linked the shared library loader that this binary relies on.

And finally I run code :

```
beeko@Nitro:~/embedded-linux-qemu-labs/toolchain$ mips-linux-gcc hello.c -o hello2
beeko@Nitro:~/embedded-linux-qemu-labs/toolchain$ file hello2
hello2: ELF 32-bit LSB executable, MIPS, MIPS-I version 1 (SYSV), dynamically linked, interpreter /lib/ld-musl-mipsel.so.1, not stripped
beeko@Nitro:~/embedded-linux-qemu-labs/toolchain$ qemu-mipsel -L '/home/beeko/x-tools/mipsel-training-linux-musl/mipsel-training-linux-musl/sysroot' hello2 Ahmed
Hello Ahmed!
```