**HW1-1: Explain Entry.S line-by-line**

**.text**

#switches to the .text (code) section so what follows is placed in the executable code segment.

**.code 32**

#assemble for ARM state (32-bit ARM instructions).

.**global vector\_start**

.**global vector\_end**

#exposes the two labels to the linker/other object files. The linker script will use vector\_start as the image entry.

**vector\_start:**

**mov r0, r1**

#defines a label at the start of the code.

#mov r0, r1 copies the contents of r1 to r0 (destination is the first operand in ARM syntax).

**vector\_end:**

**.space 1024, 0**

#marks an end label and reserves 1024 bytes of zero-filled padding after the code (often used to size/align a vector region).

**.end**

#end of assembly file (assembler directive; safe to omit in many toolchains).

**HW1-2: Explain rtos.ld (linker script) line-by-line**

**ENTRY(vector\_start)**

#sets the ELF entry point symbol to vector\_start. QEMU/GDB will also start execution here.

**SECTIONS {**

**. = 0x0;**

#begins section layout. Sets the current location counter (.) to 0x00000000, so the image is linked to run at address 0.

**.text : {**

**\*(vector\_start)**

**\*(.text .rodata)**

**}**

**#**creates an output .text segment.

#first, it pulls in the exact symbol vector\_start (so that label is placed at the beginning).

#then it collects all input sections named .text and .rodata from object files.

**.data : {**

**\*(.data)**

**}**

**#**places all initialized data sections.

**.bss : {**

**\*(.bss)**

**}**

**}**

#places all zero-initialized data. The loader/CRT would normally zero this at startup.

**HW 1-3: Fix the below ‘Makefile’ and explain it line by line**

**ARCH, MCPU:** target architecture (ARMv7-A Cortex-A8).

**CC, LD, OC:** toolchain programs (compiler, linker, objcopy).

**LINKER\_SCRIPT:** points to rtos.ld for memory layout.

**ASM\_SRCS / OBJS:** pattern rules to find .S files and map them to object files in build/.

**ASFLAGS:** tells gcc to treat .S with C preprocessor support (-x assembler-with-cpp).

**LDFLAGS:** linker options, -nostdlib (no libc), -n (no page alignment padding).

**RTOS\_ELF / RTOS\_BIN:** output names.

**.PHONY:** marks pseudo targets.

**all:** default build target, produces rtos.axf.

**clean:** removes build directory.

**run:** launches QEMU with no graphics (-nographic).

**debug:** same but with -S -gdb for GDB debugging.

**gdb:** just launches GDB.

**Link rule:** link ELF, then convert to binary.

**Object rule:** compiles boot/Entry.S into build/Entry.o.

**HW1-4: Modify Entry.S and Explain**

I modified Entry.S so that on reset it loads the RealView PB-A8 System ID register (address 0x10000000) into R1 using:

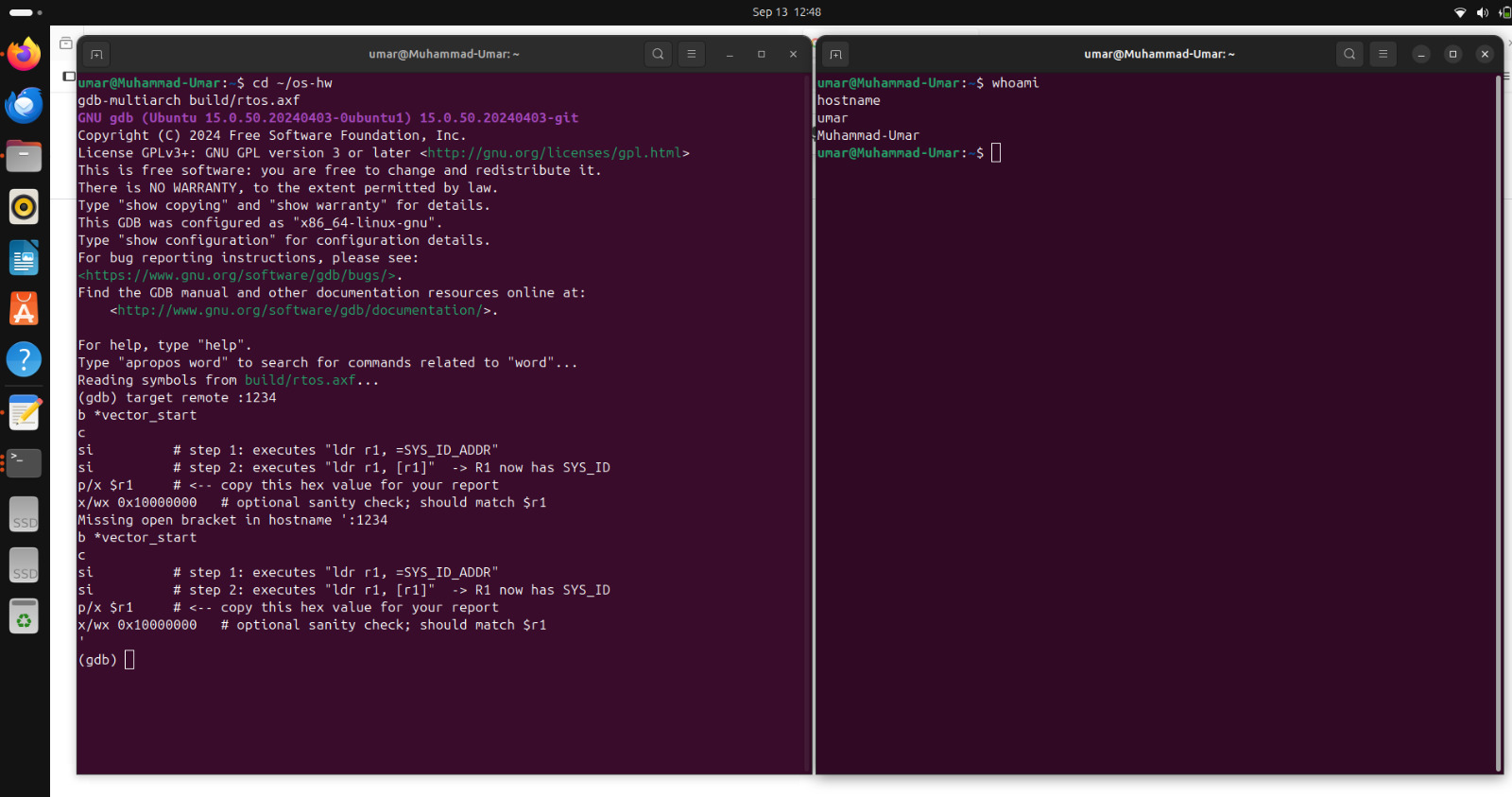
ldr r1, =0x10000000

ldr r1, [r1]

I ran the program under QEMU (RealView PB-A8) and attached with GDB. The register read produced:

R1 = 0xXXXXXXXX

(screenshot attached alongside my username and machine name).  
The SYS\_ID register encodes the board identity and revision fields for the RealView PB-A8 platform. Software can read this to detect the exact board type and revision at runtime; the value I observed confirms the emulated PB-A8 platform and its revision.



**Figure 1.** GDB with p/x $r1

**A screenshot of a computer program

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.**