CS 310 Execution Level Lab
Spring 2021

February 23, 2021

1 Privileged Operations and Execution Level

Basically all microprocessors implement some form of privilege hierarchy for software that they run. The simplest form of privilege separation is a binary distinction between userland (unprivileged) and operating system (privileged). Userland code (regular program binaries) run in unprivileged mode with no direct access to I/O devices or memory outside of their own process address space. This can prevent buggy or malicious code from overwriting important data structures and crashing the system. The ARM Cortex-A8 extends the concept a bit further, implementing four privilege levels which it calls *execution levels*.

`EL0`	Lowest privilege level. Userland code.
`EL1`	Operating system.
`EL2`	Hypervisor
`EL3`	Highest privilege level, used by TrustZone (ARM's programmable trusted computing state).

A special architectural register called the `CurrentEL` register stores the current execution level that the processor is executing in. You can use the `mrs` and `msr`[1] instructions to read and write `CurrentEL` respectively. For example, to get the current execution level into variable `el`, we can do:

```
unsigned int el;

asm("mrs %0,CurrentEL"
    : "=r"(el)
    :
    :);
```

Task 1: write a function in assembly language that reads and returns the `CurrentEL` register.

2 Transitioning to a Lower Execution Level

When the CPU boots, it starts in EL3, and the OS code must transition to EL1. Before reducing the privilege levels, we need to do some configuration. There are lots of other special-purpose registers in the ARM microprocessor that can be accessed with `mrs` and `msr`.

Task 2: There is some pre-made code on the course website to transition your CPU to EL1. Copy-paste that code in to `boot.s` and run it. Check the output of your CurrentEL function to make sure that you are now in EL1.

Task 3: Modify the code you copy-pasted into `boot.s` to transition your processor to EL0 instead of EL1. This is the level where userland programs run. Try calling one of your driver functions (LED initialization or serial port) from EL0. The computer should take a dump.

[1] `mrs` stands for move **`to`** register from special purpose register. The name for the instruction seems kind of backwards.

3 System Register Details

If you're curious, there are some details about how to set up the system registers for transitioning the processor into EL1.

3.1 `SCTLR`

The Cortex-A8 CPU lets us configure settings individually for each execution level by writing config values to the System Control Register (called `SCTLR_ELx` for execution level x). Complete description of this register can be found in Section D10.2.100 (page D10-2654) of the AArch64 Architecture Reference Manual.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EnIA	EnIB	LSMAOE	nTLSMD	EnDA	UCI	EE	E0E	SPAN	RES1	IESB	RES1	WXN	nTWE	RES0	nTWI

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
UCT	DZE	EnDB	I	RES1	RES0	UMA	SED	ITD	RES0	CP15BEN	SA0	SA	C	A	M

Bit Number	**Set To**	**Name**	**Description**
31	0	`EnIA`	Enable (1) /disable (0) pointer authentication in EL1
30	0	`EnIB`	Enable (1) /disable (0) pointer authentication in EL1
29	1	`LSMAOE`	Load/store multiple atomicity enable
28	1	`nTLSMD`	Unimplemented on RPI. Set to 1.
27	0	`EnDA`	Enable (1) /disable (0) pointer authentication in EL1
26	0	`UCI`	Trap cache maintenance instructions to EL1
25	0	`EE`	Endianness of data accesses at EL1 (0 is little endian, 1 is big)
24	0	`E0E`	Endianness of data accesses at EL0 (0 is little endian, 1 is big)
23	1	`SPAN`	Set Privilege Access Never on taking an exception to EL1
22	1	`RES1`	Reserved, set to 1
21	0	`IESB`	Implicit error synchronization event enable.
20	1	`RES1`	Reserved, set to 1
19	0	`WXN`	If set, all writable regions of memory forced execute never in EL0 and EL1.
18	1	`nTWE`	Traps `WFE` instructions at EL0 to EL1.
17	0	`RES0`	Reserved, set to 0
16	1	`nTWI`	Traps `WFI` instructions at EL0 to EL1.
15	0	`UCT`	Traps EL0 accesses to the CTR_EL0 to EL1, from AArch64 state only.
14	0	`DZE`	Traps EL0 execution of DC ZVA instructions to EL1, from AArch64 state only.
13	0	`EnDB`	Enable pointer authentication of instruction addresses in the EL1&0.
12	0	`I`	Instruction cachability for accesses at EL0 and EL1
11	1	`RES1`	Reserved, set to 1
10	0	`RES0`	Reserved, set to 0
9	0	`UMA`	User Mask Access. Traps EL0 execution of MSR and MRS instructions.
8	0	`SED`	SETEND instruction disable. Disables SETEND instructions at EL0 using AArch32.
7	0	`ITD`	IT disable: disables IT instructions in AArch32 mode.
6	0	`RES0`	Reserved, set to 0
5	1	`CP15BEN`	Instruction memory barrier enable. Enables DMB, DSB, and ISB instructions in EL0
4	0	`SA0`	Stack pointer alignment check enable for EL0.
3	0	`SA`	Stack pointer alignment check enable for EL1.
2	1	`C`	Cachability control
1	0	`A`	Alignment check enable
0	0	`M`	MMU Enable

3.2 `HCR`

The Hypervisor Configuration Register configures a bunch of stuff for virtualization. The main thing we will use it for is to configure the CPU to run in 64-bit mode by setting bit 31 to 1.

```
ldr x0, =0x80000000
msr hcr_el2, x0
```

3.3 SCR

The Secure Configuration Register controls the security state of the CPU. We will use it to set the CPU to an unsecured state for now by setting bit 0 to 1. We will also set bit 10 to 1 to enable 64-bit mode in EL0. Bits 5 and 4 are reserved and should be set to 1. Detailed description of the `SCR` is in Section D10.2.99 of the AArch64 Manual.

```
ldr x0, =0x431
msr scr_el3, x0
```

3.4 SPSR

The Saved Program Status Register saves the program state when the CPU transitions from a lower execution level to a higher one (for example to handle an interrupt). The state from the `SPSR` is then restored when the CPU transitions back to the lower execution level.