The Async Controller specification

The overall design is specified in design.

Goal

1. Achieving light weight task scheduling and switching.
2. Reducing the behavior of crossing address space and the cost of each crossing.

Registers

eptr

```
31                        0
+-------------------------+
| the pointer of Executor |
+-------------------------+
```

`Executor` is stored in the physical memory. So this register will record the pointer of `Executor` of the active address space.

status

```
31     30                     0
+------+---------------------+
| mode | the number of cause |
+------+---------------------+
```

`mode`:

- yield: finish, await
- exception
- interrupt

`code`: This field is used to record the cause of control flow changing.

- If the control flow is actively yield, the code field will be ignored.
- In there are interrupts/exceptions, the code is the same as the `Exception Code` field in the `scause` register

curc

```
31                                  0
+-----------------------------------+
| the pointer of current coroutine |
+-----------------------------------+
```

When syscall/interrupt/exception occuring, we must pass the current coroutine pointer to the related kernel task(function) according to this register.

msgbuf

```
31                                    0
+-------------------------------------+
| the base address of Message Buffer |
+-------------------------------------+
```

This register will record the base address of message buffer which is in physical memory.

keptr

```
31                               0
+--------------------------------+
| the pointer of kernel Executor |
+--------------------------------+
```

When trap into kernel, the controller will change the `eptr` register according to this one.

Task Scheduling and Switching

The controller must read the `Executor` from physical memory according to `eptr` register.

The structure of `Executor` is shown below.

```
// Executor
+-----------------+---------+---------+--------+---------+
| Priority Bitmap | Queue 1 | Queue 2 | ...... | Queue N |
+-----------------+---------+---------+--------+---------+
// Queue
+------------------+------------------+------------------+
| Slots of TaskRef | Dequeue position | Enqueue position |
+------------------+------------------+------------------+
```

Discussion: The `Executor` can be implemented in the hardware by using the exist IP cores(e.g., FIFO). When changing address space, the data mover help read/flush `Executor` from/to the physical memory.

Message Communication

The communication is between the kernel and user application.

- If there are multiple cores, it's actually between the application in one core and the kernel in the other core.
- If there is only one core, it's between the application and kernel in the same core. If there are ready tasks, the control flow will not trap into kernel. So the syscall/interrupt/exception will be batched.

Discussion: Maybe it can be implemented by using FIFO queue within the controller instead of using shared memory.

The message buffer will store message entries. The implementation of each message entry is shown below.

```
high                                                       low
+---------------------+---------------------------+-----------+
| address space token | related user task pointer | arguments |
+---------------------+---------------------------+-----------+
```

The address space token must be recorded due to the kernel syscall task may read/write data in the application address space.

The related user task pointer must be recorded due to the kernel syscall task may need to wake the user task.

`arguments`:

- syscall:

```rust
pub struct SyscallArg {
    id: usize,
    a: [usize; 7]
}
```

- interrupt:

```rust
pub struct InterruptArg {
    interrupt: Interrupt,
}
```

- exception:

```rust
pub struct ExceptionArg {
    exception: Exception,
    stval: usize
}
```

Interface

- init： this function will only be used in kernel. It will fill the `eptr` and `msgbuf` register.
- get_curc: Supporting to query the currently running coroutine.
- fetch_message: kernel will fetch the message entry by this function.
-