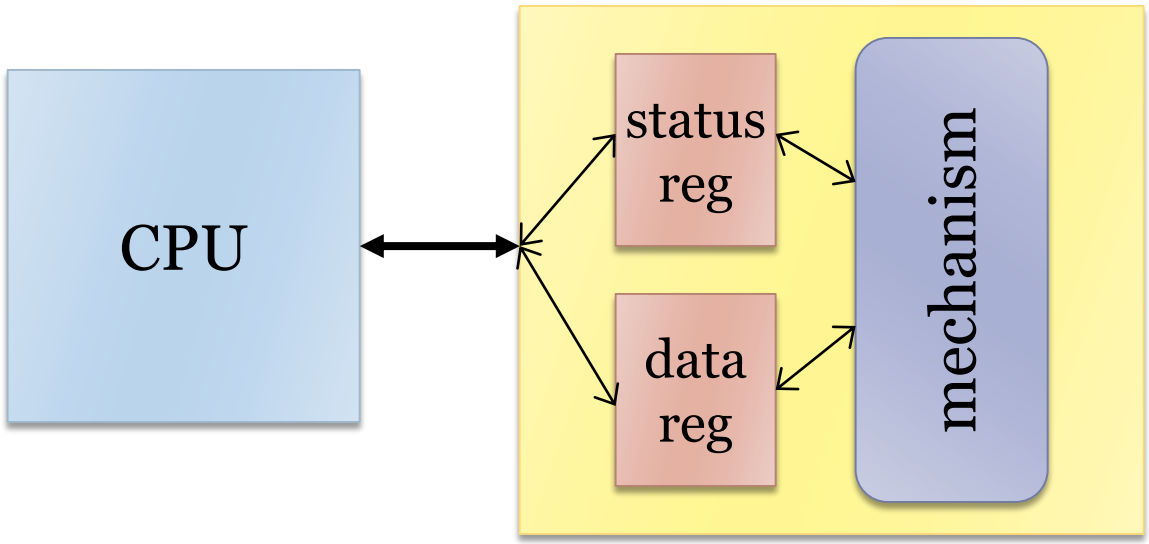
Input/Output

Jin-Soo Kim (*jinsookim@skku.edu*)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu

I/O Devices

- Usually includes some non-digital component
- Typical digital interface to CPU:

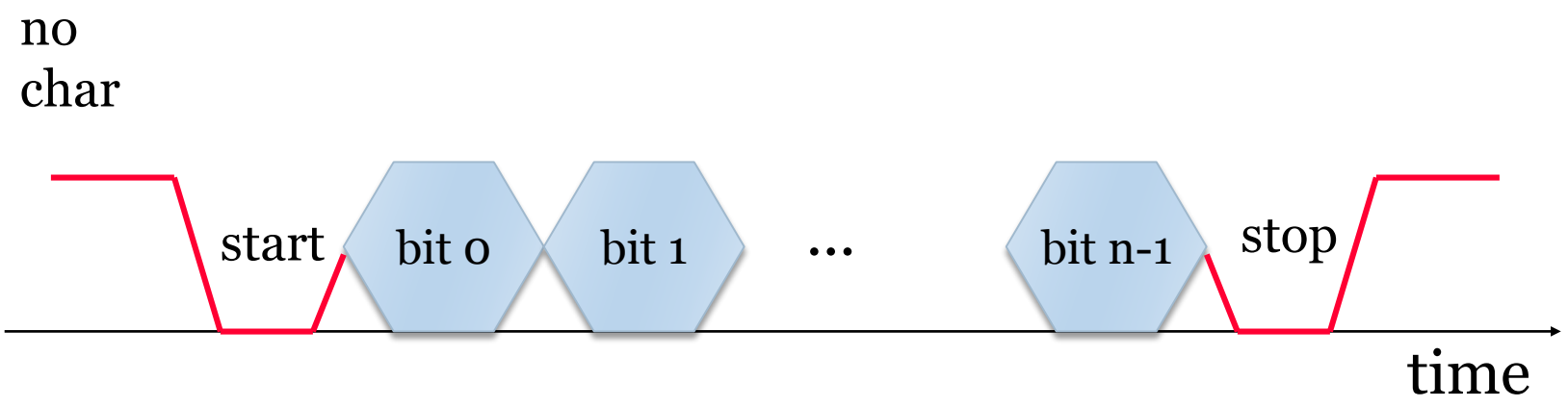


8251 UART

- Universal asynchronous receiver/transmitter (UART)
- Provides serial communication
- 8251 functions are integrated into standard PC interface chip
- Allows many communication parameters to be programmed

Serial Communication

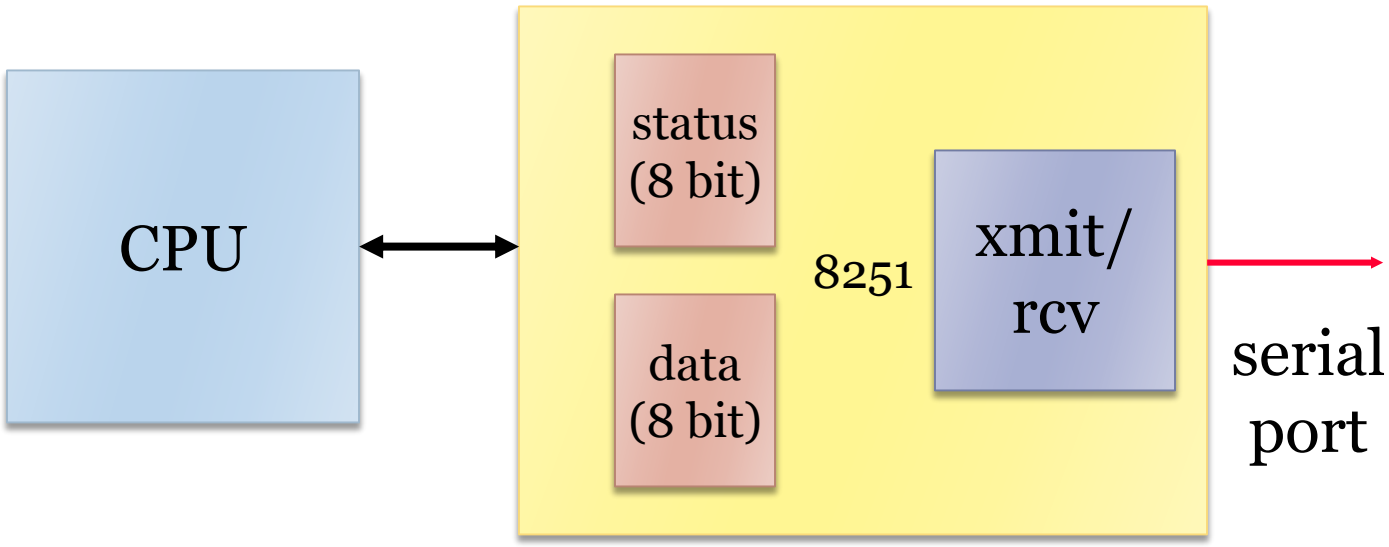
- Characters are transmitted separately:



Serial Comm. Parameters

- Baud (bit) rate
  - 50, 300, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600, 115200 bps
- Number of bits per character
  - 5, 6, 7, 8 bits
- Parity/no parity
- Even/odd parity
- Length of stop bit
  - 1, 1.5, 2 bits

8251 CPU Interface



Programming I/O

- Two types of instructions can support I/O:
  - Special-purpose I/O instructions
  - Memory-mapped load/store instructions
- Intel x86 provides in, out instructions. Most other CPUs use memory-mapped I/O
- I/O instructions do not preclude memory-mapped I/O

ARM Memory-Mapped I/O

- Define location for device (mem address)

```
DEV1    EQU     0x1000
```

- Read/write code

```
LDR     r1, =DEV1  ; set up device address
LDR     r0, [r1]   ; read DEV1
MOV     r0, #8     ; set up value to write
STR     r0, [r1]   ; write value to device
```

Peek and Poke

- Traditional HLL interfaces:

```
int peek (char *location)
{
        return *location;
}

int poke (char *location, char newval)
{
        (*location) = newval;
}
```

Busy-Wait Output

- Simplest way to program device
  - Use instructions to test when device is ready

```
current_char = mystring;
while (*current_char != '\0')  {
    while (peek (OUT_STATUS) != 0);
    poke (OUT_CHAR, *current_char);
    current_char++;
}
```

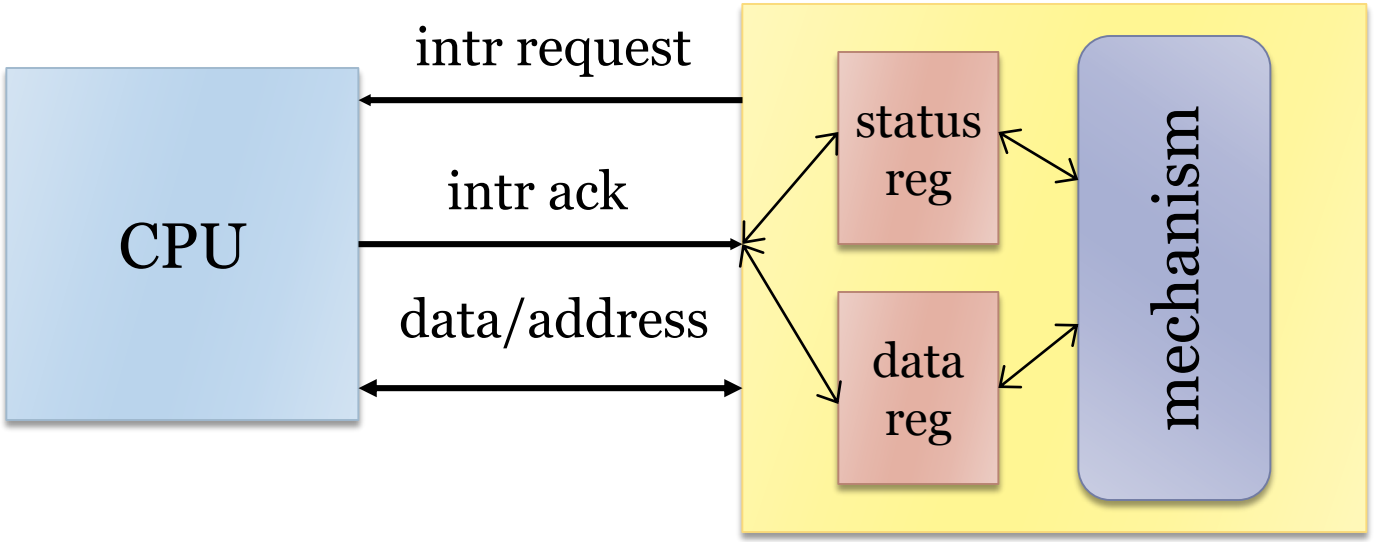
The output status register's value is 1 when the device is busy writing

Interrupt I/O

- Busy-wait (or *polling*) is very inefficient
  - CPU can't do other work while testing device
  - Hard to do simultaneous I/O

- Interrupts allow a device to change the flow of control in the CPU
  - Causes subroutine call to handle device

Interrupt Interface



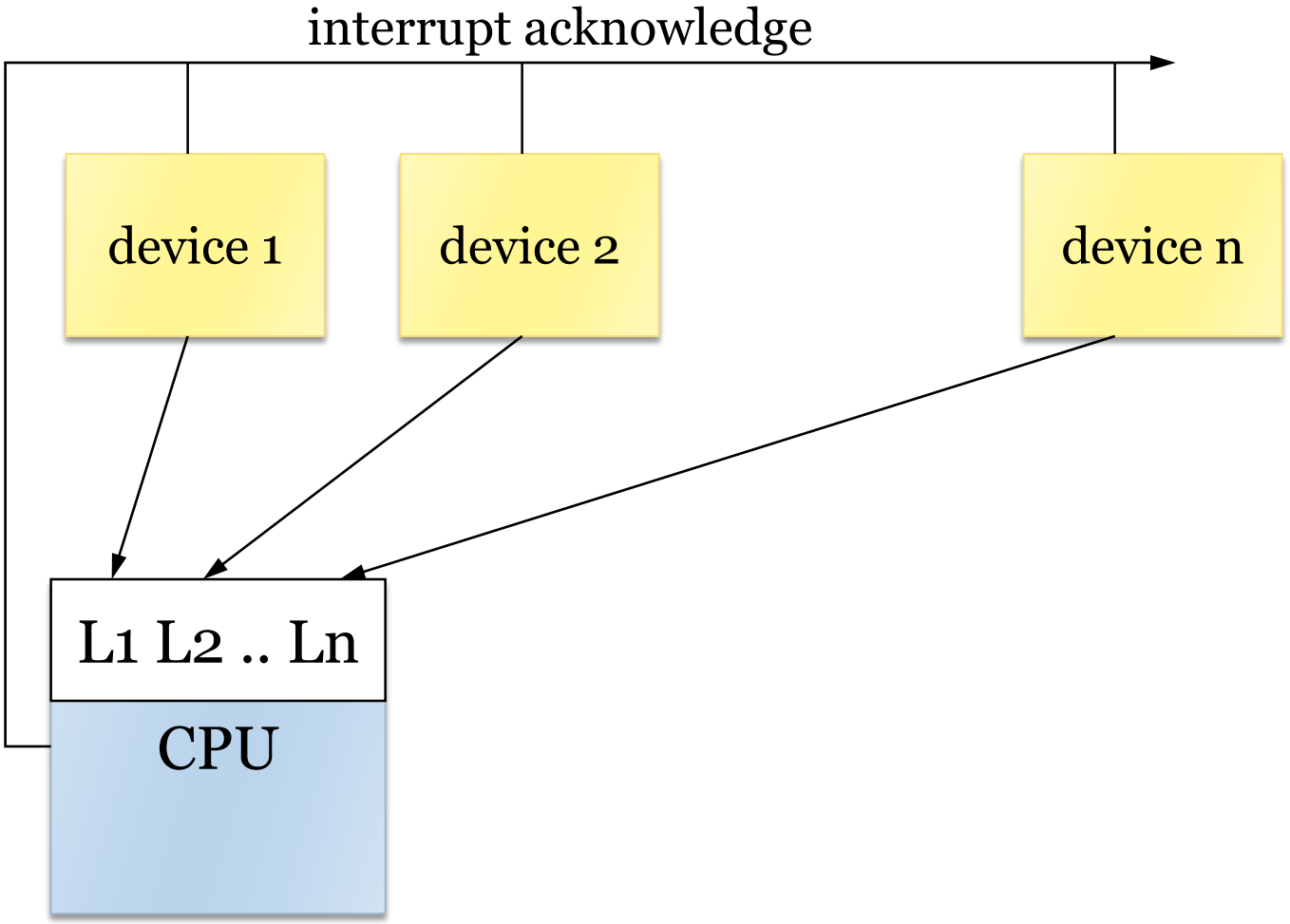
Interrupt Behavior

- Based on subroutine call mechanism
- Device asserts interrupt request
- CPU asserts interrupt acknowledge when it can handle the interrupt
- Interrupt forces next instruction to be a subroutine call to a predetermined location
- Return address is saved to resume executing foreground program

Priorities and Vectors

- Two mechanisms allow us to make interrupts more specific:
  - Priorities determine what interrupt gets CPU first
  - Vectors determine what code is called for each type of interrupt
- Mechanisms are orthogonal: most CPUs provide both

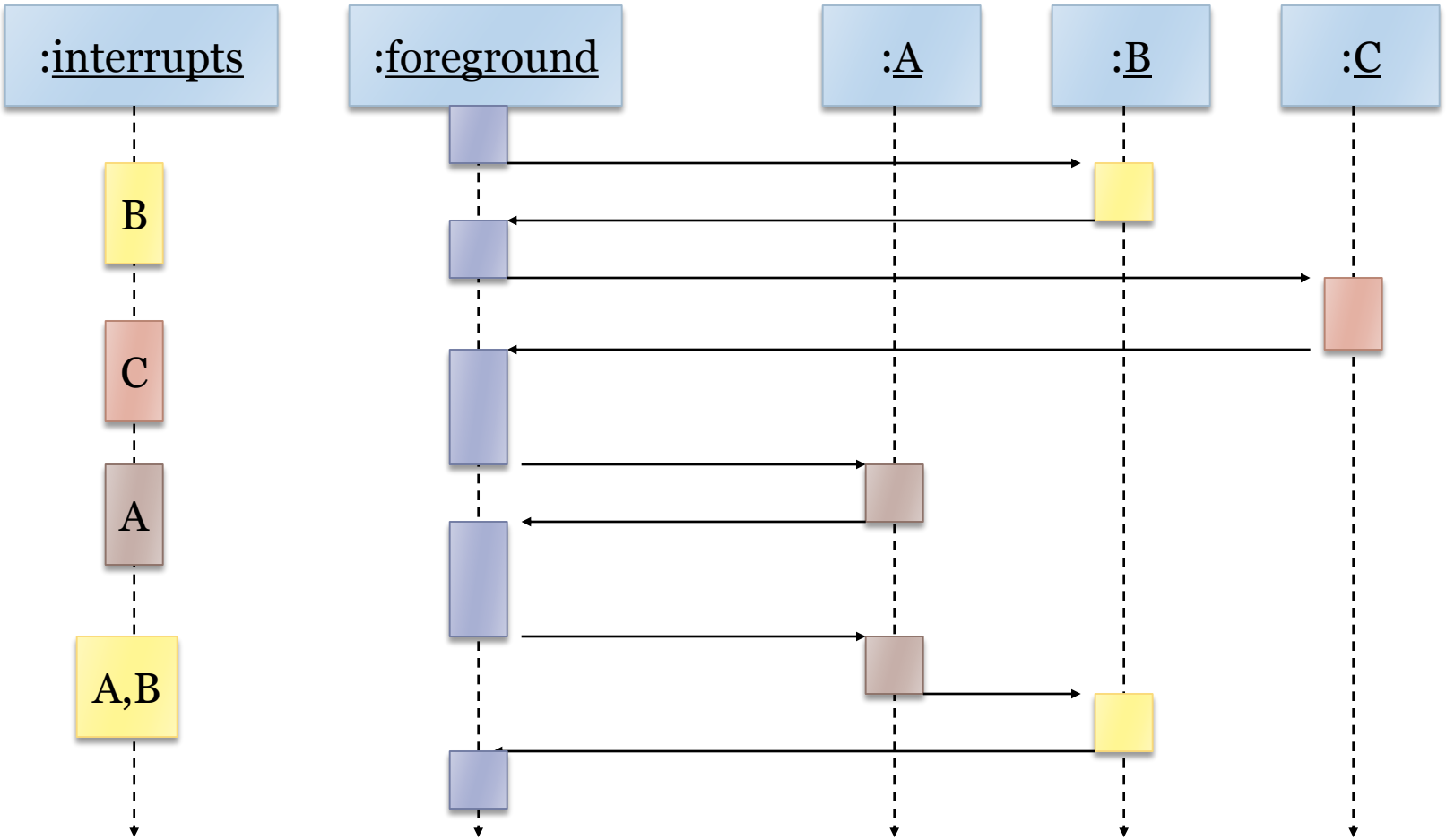
Prioritized Interrupts



Interrupt Prioritization

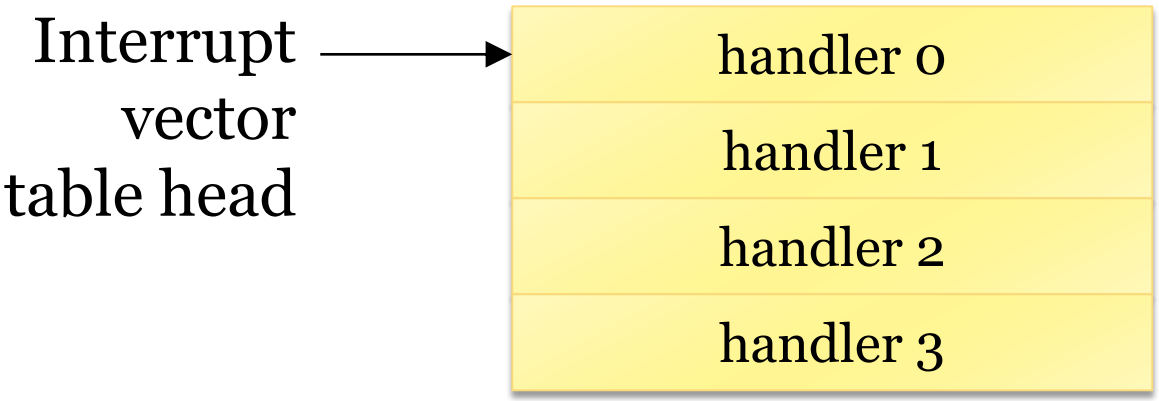
- Masking
  - Interrupt with priority lower than current priority is not recognized until pending interrupt is complete
- Non-maskable interrupt (NMI)
  - Highest-priority, never masked
  - Often used for power-down

Example: Prioritized I/O



Interrupt Vectors

- Allow different devices to be handled by different code
- Interrupt vector table:



Interrupt Sequence

- CPU acknowledges request
- Device sends vector
- CPU calls handler
- Software processes request
- CPU restores state to foreground program

Interrupt Overhead

- Handler execution time
- Interrupt mechanism overhead
  - Acknowledging the interrupt
  - Obtaining the vector from the device
- Register save/restore
- Pipeline-related penalties
- Cache-related penalties

ARM Exceptions

Types of Exception

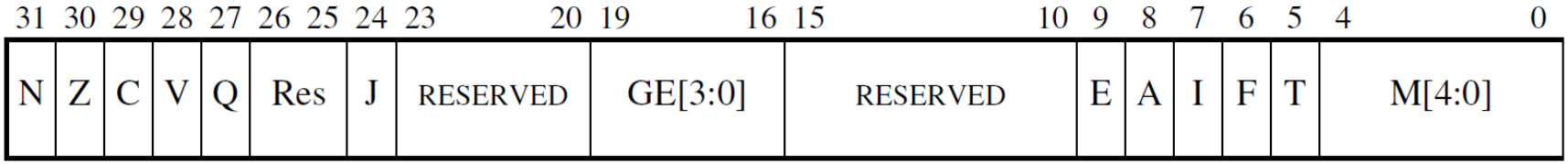
- Reset
- Undefined instruction
- Software interrupt (SWI)
- Prefetch abort
- Data abort
- IRQ (normal interrupt)
- FIQ (Fast interrupt)

Processor Modes

- Mode changes by software control, or by external interrupts or exception processing

Processor mode		Mode number	Description
User	usr	0b10000	Normal program execution mode
FIQ	fiq	0b10001	Supports a high-speed data transfer or channel process
IRQ	irq	0b10010	Used for general-purpose interrupt handling
Supervisor	svc	0b10011	A protected mode for the operating system
Abort	abt	0b10111	Implements virtual memory and/or memory protection
Undefined	und	0b11011	Supports software emulation of hardware coprocessors
System	sys	0b11111	Runs privileged operating system tasks (ARMv4 and above)

CPSR



- M[4:0]: The mode bits
- T: ARM(0), Thumb(1)
- F: Enable(0) or disable(1) FIQ interrupts
- I: Enable(0) or disable(1) IRQ interrupts

Exception Processing Modes

Exception type	Mode	Normal address	High vector address
Reset	Supervisor	0x00000000	0xFFFF0000
Undefined instructions	Undefined	0x00000004	0xFFFF0004
Software interrupt (SWI)	Supervisor	0x00000008	0xFFFF0008
Prefetch Abort (instruction fetch memory abort)	Abort	0x0000000C	0xFFFF000C
Data Abort (data access memory abort)	Abort	0x00000010	0xFFFF0010
IRQ (interrupt)	IRQ	0x00000018	0xFFFF0018
FIQ (fast interrupt)	FIQ	0x0000001C	0xFFFF001C

Vector address 0x00000014/0xffff0014 are reserved.
The bit 13 of the System control coprocessor (CP15) register 1 selects the location of the exception vectors

Co-processor

- Co-processor: added function unit that is called by instruction
  - Floating-point units are often structured as co-processors
- ARM allows up to 16 designer-selected co-processors
  - Floating-point co-processor uses units 1, 2
  - System control processor 15

Exception Priorities

Priority		Exception
Highest	1	Reset
	2	Data Abort (including data TLB miss)
	3	FIQ
	4	IRQ
	5	Imprecise Abort (external abort) - ARMv6
	6	Prefetch Abort (including prefetch TLB miss)
Lowest	7	Undefined instruction SWI

Register Organization

	Privileged modes						
		Exception modes					
	User	System	Supervisor	Abort	Undefined	Interrupt	Fast interrupt
	R0	R0	R0	R0	R0	R0	R0
	R1	R1	R1	R1	R1	R1	R1
	R2	R2	R2	R2	R2	R2	R2
	R3	R3	R3	R3	R3	R3	R3
	R4	R4	R4	R4	R4	R4	R4
	R5	R5	R5	R5	R5	R5	R5
	R6	R6	R6	R6	R6	R6	R6
	R7	R7	R7	R7	R7	R7	R7
	R8	R8	R8	R8	R8	R8	R8_fiq
	R9	R9	R9	R9	R9	R9	R9_fiq
	R10	R10	R10	R10	R10	R10	R10_fiq
	R11	R11	R11	R11	R11	R11	R11_fiq
	R12	R12	R12	R12	R12	R12	R12_fiq
SP	R13	R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
LR	R14	R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
	PC	PC	PC	PC	PC	PC	PC
	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
			SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

IRQ

- On entry

```
R14_irq      = address of next instruction to be executed + 4
SPSR_irq     = CPSR
CPSR[4:0]    = 10010          /* IRQ mode */
CPSR[5]      = 0              /* ARM state */
CPSR[7]      = 1              /* disable normal interrupts */
PC           = 0x00000018
```

- On exit (by programmer)

```
SUBS    PC, R14, #4          /* SPSR is moved to CPSR */
```

FIQ

- On entry

```
R14_fiq      = address of next instruction to be executed + 4
SPSR_fiq     = CPSR
CPSR[4:0]    = 10001          /* FIQ mode */
CPSR[5]      = 0              /* ARM state */
CPSR[6]      = 1              /* disable fast  interrupts */
CPSR[7]      = 1              /* disable normal interrupts */
PC           = 0x0000001C
```

- On exit (by programmer)

```
SUBS    PC, R14, #4          /* SPSR is moved to CPSR */
```

ARM Supervisor Mode

- Use SWI instruction to enter supervisor mode, similar to subroutine:
  - SWI CODE_1
- Sets PC to 0x08
- Arguments to SWI is passed to supervisor mode code
- Saves CPSR in SPSR

ARM Interrupt Latency

- Worst-case latency to respond to interrupt is 27 cycles:
  - Two cycles to synchronize external request
  - Up to 20 cycles to complete current instruction
  - Three cycles for data abort
  - Two cycles to enter interrupt handling state
- Best-case latency is 4 cycles.

Jasmine OpenSSD Code

Entry Point

- @ ./target_spw/init_rvds.s

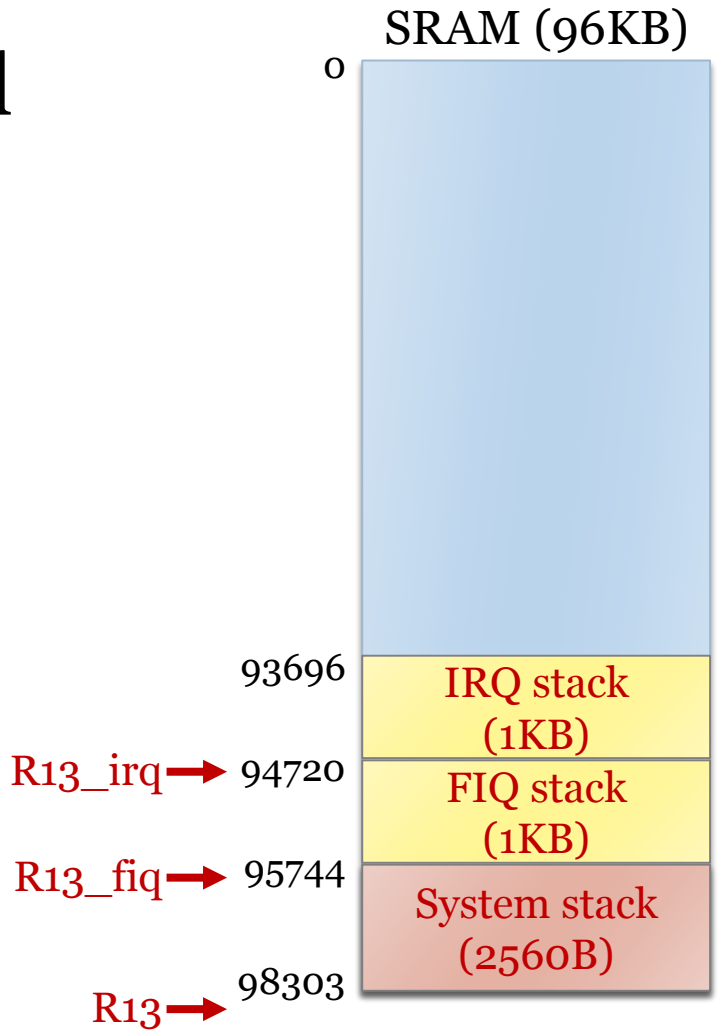
```
MODE_USR        EQU     0x10
MODE_FIQ        EQU     0x11
MODE_IRQ        EQU     0x12
MODE_SVC        EQU     0x13
MODE_ABT        EQU     0x17
MODE_UND        EQU     0x1B
MODE_SYS        EQU     0x1F

I_BIT           EQU     0x80
F_BIT           EQU     0x40

        PRESERVE8
        AREA    init, CODE, READONLY

        ENTRY

        B reset_handler ; reset
        B .             ; undefined instruction
        B .             ; SWI
        B .             ; prefetch abort
        B .             ; data abort
        NOP             ; reserved vector
        B irq_handler   ; IRQ
        B fiq_handler   ; FIQ
```

```
reset_handler

        ; IRQ mode stack

        MSR         CPSR_c, #MODE_IRQ:OR:I_BIT:OR:F_BIT
        LDR         R13, =|Image$$ER_IRQ_STACK$$ZI$$Limit|

        ; FIQ mode stack

        MSR         CPSR_c, #MODE_FIQ:OR:I_BIT:OR:F_BIT
        LDR         R13, =|Image$$ER_FIQ_STACK$$ZI$$Limit|

        ; SYSTEM mode stack
        ; SYSTEM mode is the main mode of Barefoot firmware.

        MSR         CPSR_c, #MODE_SYS:OR:I_BIT:OR:F_BIT
        LDR         R13, =|Image$$ER_SYS_STACK$$ZI$$Limit|

        BL          init_jasmine
        BL          Main
        B           .       ; should not reach here
```

Stacks

@ ./build_rvds/scatter.scl

```
LR_IMAGE 0x00000000          96*1024
{
    ER_CODE                  0x00000000
    {    init.o (init, +FIRST)
         *.o (+RO)                        }
    ER_RW +0
    {    * (+RW)                          }
    ER_ZI +0
    {    * (+ZI)                          }
    ER_IRQ_STACK   93696     EMPTY     1024
    {                                     }
    ER_FIQ_STACK   94720     EMPTY     1024
    {                                     }
    ER_SYS_STACK   95744     EMPTY     2560
    {                                     }
}
```



Disabling/Enabling Interrupts

- @ ./target_spw/init_rvds.s

```
disable_interrupt

        MRS         R0, CPSR
        ORR         R0, R0, #0xC0
        MSR         CPSR_c, R0
        BX          LR

enable_interrupt

        MRS         R0, CPSR
        BIC         R0, R0, #0xC0
        MSR         CPSR_c, R0
        BX          LR
```