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:

```
no char
```

```
start bit 0 bit 1 ... bit n-1 stop
```

```
time
```
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)

```assembly
DEV1 EQU 0x1000
```

- Read/write code

```assembly
LDR r1, =DEV1 ; set up device address
LDR ro, [r1] ; read DEV1
MOV ro, #8 ; set up value to write
STR ro, [r1] ; write value to device
```
Peek and Poke

- Traditional HLL interfaces:

```c
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

```cpp
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

CPU

intr request
intr ack
data/address

status reg
data reg

mechanism
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 acknowledge

device 1
device 2
device n

L1 L2 .. Ln

CPU
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

Example: Prioritized I/O

<table>
<thead>
<tr>
<th>:interrupts</th>
<th>:foreground</th>
<th>:A</th>
<th>:B</th>
<th>:C</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A, B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Interrupt Vectors

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

```
<table>
<thead>
<tr>
<th>Interrupt vector table head</th>
</tr>
</thead>
<tbody>
<tr>
<td>handler 0</td>
</tr>
<tr>
<td>handler 1</td>
</tr>
<tr>
<td>handler 2</td>
</tr>
<tr>
<td>handler 3</td>
</tr>
</tbody>
</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

<table>
<thead>
<tr>
<th>Processor mode</th>
<th>Mode number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>usr</td>
<td>0b10000</td>
</tr>
<tr>
<td>FIQ</td>
<td>fiq</td>
<td>0b10001</td>
</tr>
<tr>
<td>IRQ</td>
<td>irq</td>
<td>0b10010</td>
</tr>
<tr>
<td>Supervisor</td>
<td>svc</td>
<td>0b10011</td>
</tr>
<tr>
<td>Abort</td>
<td>abt</td>
<td>0b10111</td>
</tr>
<tr>
<td>Undefined</td>
<td>und</td>
<td>0b11011</td>
</tr>
<tr>
<td>System</td>
<td>sys</td>
<td>0b11111</td>
</tr>
</tbody>
</table>
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

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

Vector address 0x00000014/0xfffff0014 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

<table>
<thead>
<tr>
<th>Priority</th>
<th>Exception</th>
</tr>
</thead>
<tbody>
<tr>
<td>Highest</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Reset</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Data Abort (including data TLB miss)</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>FIQ</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>IRQ</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Imprecise Abort (external abort) - ARMv6</td>
</tr>
<tr>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Prefetch Abort (including prefetch TLB miss)</td>
</tr>
<tr>
<td>Lowest</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Undefined instruction</td>
</tr>
<tr>
<td></td>
<td>SWI</td>
</tr>
</tbody>
</table>
Register Organization

<table>
<thead>
<tr>
<th>User</th>
<th>System</th>
<th>Supervisor</th>
<th>Abort</th>
<th>Undefined</th>
<th>Interrupt</th>
<th>Fast interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>R0</td>
<td>R0</td>
<td>R0</td>
<td>R0</td>
<td>R0</td>
<td>R0</td>
</tr>
<tr>
<td>R1</td>
<td>R1</td>
<td>R1</td>
<td>R1</td>
<td>R1</td>
<td>R1</td>
<td>R1</td>
</tr>
<tr>
<td>R2</td>
<td>R2</td>
<td>R2</td>
<td>R2</td>
<td>R2</td>
<td>R2</td>
<td>R2</td>
</tr>
<tr>
<td>R3</td>
<td>R3</td>
<td>R3</td>
<td>R3</td>
<td>R3</td>
<td>R3</td>
<td>R3</td>
</tr>
<tr>
<td>R4</td>
<td>R4</td>
<td>R4</td>
<td>R4</td>
<td>R4</td>
<td>R4</td>
<td>R4</td>
</tr>
<tr>
<td>R5</td>
<td>R5</td>
<td>R5</td>
<td>R5</td>
<td>R5</td>
<td>R5</td>
<td>R5</td>
</tr>
<tr>
<td>R6</td>
<td>R6</td>
<td>R6</td>
<td>R6</td>
<td>R6</td>
<td>R6</td>
<td>R6</td>
</tr>
<tr>
<td>R7</td>
<td>R7</td>
<td>R7</td>
<td>R7</td>
<td>R7</td>
<td>R7</td>
<td>R7</td>
</tr>
<tr>
<td>R8</td>
<td>R8</td>
<td>R8</td>
<td>R8</td>
<td>R8</td>
<td>R8</td>
<td>R8</td>
</tr>
<tr>
<td>R9</td>
<td>R9</td>
<td>R9</td>
<td>R9</td>
<td>R9</td>
<td>R9</td>
<td>R9</td>
</tr>
<tr>
<td>R10</td>
<td>R10</td>
<td>R10</td>
<td>R10</td>
<td>R10</td>
<td>R10</td>
<td>R10</td>
</tr>
<tr>
<td>R11</td>
<td>R11</td>
<td>R11</td>
<td>R11</td>
<td>R11</td>
<td>R11</td>
<td>R11</td>
</tr>
<tr>
<td>R12</td>
<td>R12</td>
<td>R12</td>
<td>R12</td>
<td>R12</td>
<td>R12</td>
<td>R12</td>
</tr>
<tr>
<td>R13</td>
<td>R13</td>
<td>R13_svc</td>
<td>R13_abt</td>
<td>R13_svc</td>
<td>R13_abt</td>
<td>R13_svc</td>
</tr>
<tr>
<td>R14</td>
<td>R14</td>
<td>R14_svc</td>
<td>R14_abt</td>
<td>R14_svc</td>
<td>R14_abt</td>
<td>R14_svc</td>
</tr>
<tr>
<td>PC</td>
<td>PC</td>
<td>PC</td>
<td>PC</td>
<td>PC</td>
<td>PC</td>
<td>PC</td>
</tr>
<tr>
<td>CPSR</td>
<td>CPSR</td>
<td>CPSR</td>
<td>CPSR</td>
<td>CPSR</td>
<td>CPSR</td>
<td>CPSR</td>
</tr>
</tbody>
</table>

Privileged modes

Exception modes

SP
LR
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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>R14_fiq</td>
<td>address of next instruction to be executed + 4</td>
</tr>
<tr>
<td>SPSR_fiq</td>
<td>CPSR</td>
</tr>
<tr>
<td>CPSR[4:0]</td>
<td>10001</td>
</tr>
<tr>
<td>CPSR[5]</td>
<td>0</td>
</tr>
<tr>
<td>CPSR[6]</td>
<td>1</td>
</tr>
<tr>
<td>CPSR[7]</td>
<td>1</td>
</tr>
<tr>
<td>PC</td>
<td>0x0000001C</td>
</tr>
</tbody>
</table>

- 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:
  - \texttt{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

```assembly
// MODE values
MODE_USR EQU 0x10
MODE_FIQ EQU 0x11
MODE_IRQ EQU 0x12
MODE_SVC EQU 0x13
MODE_ABTS EQU 0x17
MODE_UND EQU 0x1B
MODE_SYS EQU 0x1F
I_BIT EQU 0x80
F_BIT EQU 0x40

; Entry point
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
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
    { }
}

SRAM (96KB)

System stack (2560B)

IRQ stack (1KB)

FIQ stack (1KB)
Disabling/Enabling Interrupts

- @ ./target_spw/init_rvds.s

```plaintext
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
```