Chapter 10
21. Interrupts

- Example
- Processor state
- Priority
- Interrupt-driven I/O
- Polling
- Transfer control for I/O operations

November 10, 2014
Interrupts

- Wake me when we get there.
- CPU can be performing other tasks instead of polling device.
- New data arrives OR device ready for next data
- Device sends a special signal to CPU when

Polling

- Are we there yet? Are we there yet? Are we there yet?
- New data arrives OR device ready for next data
- CPU keeps checking status register until

Review: Transfer Control

Who determines when the next data transfer occurs?
Polling and Interrupts

Asynchronous devices

<table>
<thead>
<tr>
<th>Function</th>
<th>I/O Register</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Displayed on screen.</td>
<td>Display Data Register (DDR)</td>
<td>0FE06</td>
</tr>
<tr>
<td>Character written to bits [7:0] will be displayed another char on screen.</td>
<td>Display Status Register (DSR)</td>
<td>0FE04</td>
</tr>
<tr>
<td>Bit [15] is one when device ready to type on keyboard.</td>
<td>Keyboard Data Register (KBDR)</td>
<td>0FE02</td>
</tr>
<tr>
<td>Bits [7:0] contain the last character received a new character.</td>
<td>Keyboard Status Register (KBSR)</td>
<td>0FE00</td>
</tr>
</tbody>
</table>

(Memory-mapped I/O (Table A.3))

Review: LC-3
When KBSR is read:

- Keyboard is enabled
- KBISR[15] is set to zero

When KBDR is read:

- Keyboard is disabled -- any typed characters will be ignored
- The "ready bit" (KBISR[15]) is set to one
  (bits [15:8] are always zero)
- Its ASCII code is placed in bits [7:0] of KBDR

Input from Keyboard:

Review:
Read Character

Poll? New

Poll UID RO, KDREPT

BRZP Poll

... POLL UID RO, KBDREPT

Poll UID RO, KBDREPT

Review: Basic Input Routine

Polling
Address Control Logic determines whether MDR is loaded from Memory or from KBSR/KBDR.
When DSR[15] is zero:
- Any other character data written to DDR is ignored.
- Character in DDR[7:0] is displayed.
- DSR[15] is set to zero.

When data is written to Display Data Register:

When Monitor is ready to display another character:

Review: Output to Monitor.
Review: Basic Output Routine

Poll INI DI, RL, DPR
Poll INI RO, DPR
SBP

...
Usually, input character is also printed to screen.

Review: Keyboard Echo Routine

```
STI  RO, DDRPR
BRZP POLTI2
POLTI2 I=1, RIFI, DDRPR
IDI  RO, KBDPR
BRZP POLTI1
POLTI1 I=1, RO, KBSPR
```
current input. (See Example 8.1 in text.)

Example: Process previous input while collecting
for more computation.
especially for rare events – these cycles can be used
Polling consumes a lot of cycles.

WHY?

(3) Resume the stopped program as if nothing happened.
(2) Have the processor satisfy the device's needs; and
(1) Force currently executing program to stop;

External device can:

Interrupt-Driven I/O
When ready bit is set and IE bit is set, interrupt is signaled.

- Software sets "interrupt enable" bit in device register.
- Generating signal and whether its priority is higher than the current program.
- A way for the CPU to test whether the interrupt signal is set.
- Interrupting event has occurred.
- A way for the I/O device to signal the CPU that an I/O interrupt-driven I/O is needed.
and generates interrupt signal if appropriate.

Priority encoder selects highest-priority device,

but not the other way around.

• It's OK for PLO device to interrupt PLO program,

Nuclear power correction program runs at PLO.

• Payroll program runs at PLO.

Example:

LC-3: 8 priority levels (PLO-PPL7)

Every instruction executes at a stated level of urgency.
**Priority Encoder**

![Diagram of a priority encoder]

**Truth Table**

<table>
<thead>
<tr>
<th>IN</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

* (Note: asterisk indicates a specific input combination)

<table>
<thead>
<tr>
<th>IN</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

* (Note: asterisk indicates a specific input combination)

<table>
<thead>
<tr>
<th>IN</th>
<th>10</th>
<th>00</th>
<th>00</th>
<th>00</th>
<th>00</th>
<th>00</th>
<th>00</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT</td>
<td>10</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>
Testing for Interrupt Signal

CPU looks at signal between STORE and FETCH phases.

If set, transfers control to Interrupt Service Routine.

If not set, continues with next instruction.

Details in Chapter 10

F
\(\uparrow\)
D
\(\uparrow\)
EA
\(\uparrow\)
OP
\(\uparrow\)
EX
\(\uparrow\)
S
\(\uparrow\)

 ISR
\(\uparrow\)
Transfer to

Interrupt Signal?

YES

NO

(ISR)
must be written as well as read. Because of interrupt enable bits, status registers (KBSPR/DSR)

Full Implementation of LC-3 Memory-Mapped I/O
Chapter 8 didn’t explain how (2) and (3) occur.

Because it involves a stack

Interrupt is an unscripted subroutine

triggered by an external event.

call

Interrupts were introduced in Chapter 8

Now, we're ready...

3. When finished, processor restores state and resumes program
2. Processor saves state and starts service routine
1. External device signals need to be serviced

I/O (Part 2)
All temporary state of the process that's not stored in memory.

Registers

- Pointer to next instruction to be executed.

Program Counter

- Privilege [15:8], Priority Level [7:0], Condition Codes [7:0]

Processor Status Register

What state is needed to completely capture the state of a running process?
- Push state to save, pop to restore.
- Location of stack "hard-wired".
- Use a stack!

That is, an Interrupt service routine might also get interrupted.
- Also, interrupts may be nested, so we wouldn't know where.
- Must save state before invoking routine.

Memory allocated by service routine?

- When resuming, need to restore state exactly as it was.
  - When resuming, need to restore state exactly as it was.
  - Programmer doesn't know when Interrupt might occur.

Can't use registers.

Where to save Processor State?
When switching from User mode to Supervisor mode

(as result of interrupt) save R6 to Saved.USP

Want to use R6 as stack pointer

Saved.USP

Another register for storing User Stack Pointer (USP):

Initial Supervisor Stack Pointer (SSP) stored in Saved.SSP

A special region of memory used as the stack

Supervisor Stack
Invoking the Service Routine – The Details

1. If PRIV = 1 (user),

2. Push PSR and PC to Supervisor Stack.


5. Set PSR[2:0] = 0.

6. Set MAP = x01VV, where VV = 8-bit Interrupt Vector

7. Load memory location (M[x01VV]) into MDR.

8. Set PC = MDR, now first instruction of ISR will be fetched.

Note: This all happens between

the FETCH of the first ISR instruction
and the STORE RESULT of the last user instruction and
(More about that later)

- If executed in User Mode, causes an exception
- Can only be executed in Supervisor Mode

RTR is a privileged instruction.

(If going back to user mode, need to restore User Stack Pointer.)

1. Pop PC from Supervisor stack. (PC = M[RG]; RG = RG + 1)
2. Pop PSR from Supervisor stack. (PSR = M[RG]; RG = RG + 1)
3. If PSR[15] = 1, RG = Saved. USP.

Returning from Interrupt.
Executing AND at x6202 when Device C Interrupts.

Executing ADD at location x3006 when Device B Interrupts.

Example
Example (2)
Example (3)

Program A

ISR for Device B

RTS

AND

Device

Interrupts

PC

x6203

R(6)→

PSR for A

x3007

x3006

x6202

x6200

x6210
Device C Service Routine (at x6300).

Push PSR and PC onto stack, then transfer to

Example (4)
Execute RTI at x6315; pop PC and PSR from stack.
Execute RTI at x6210; pop PSR and PC from stack.

Example (6)