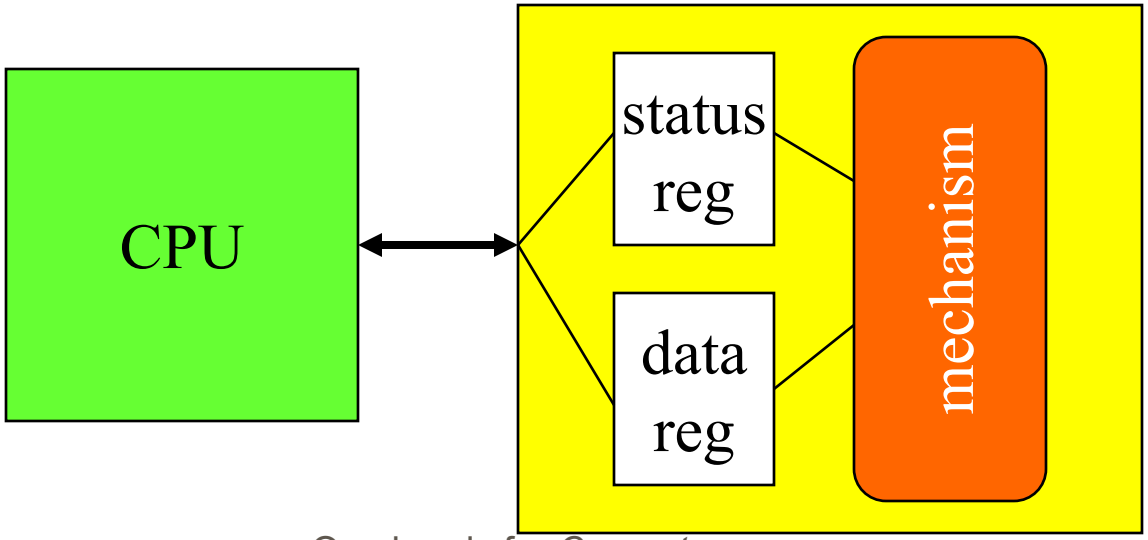
CPUs

- Input and output.
- Supervisor mode, exceptions, traps.
- Co-processors.

I/O devices

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

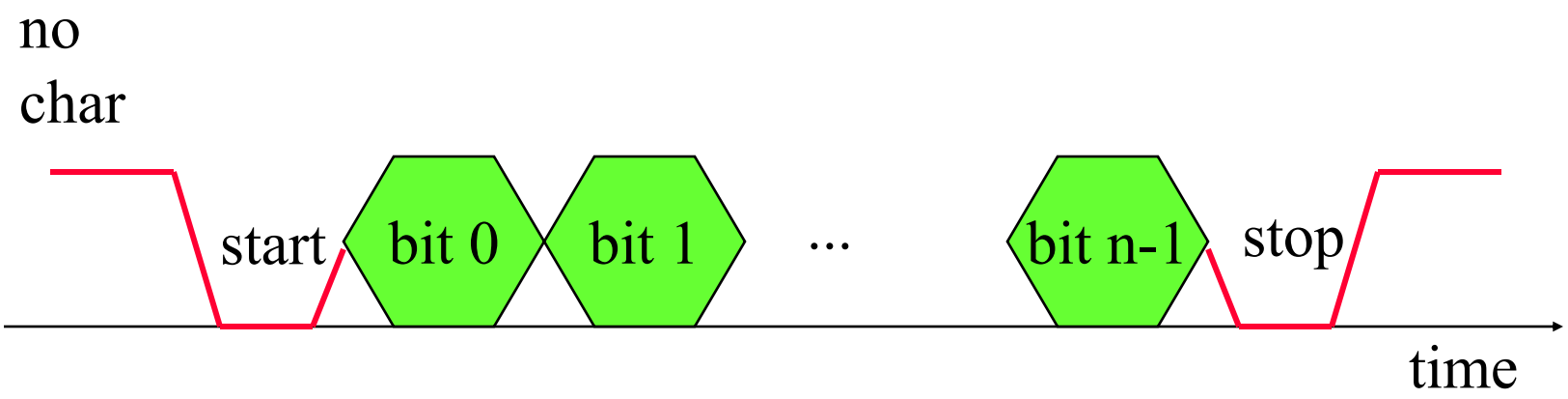


Application: 8251 UART

- 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

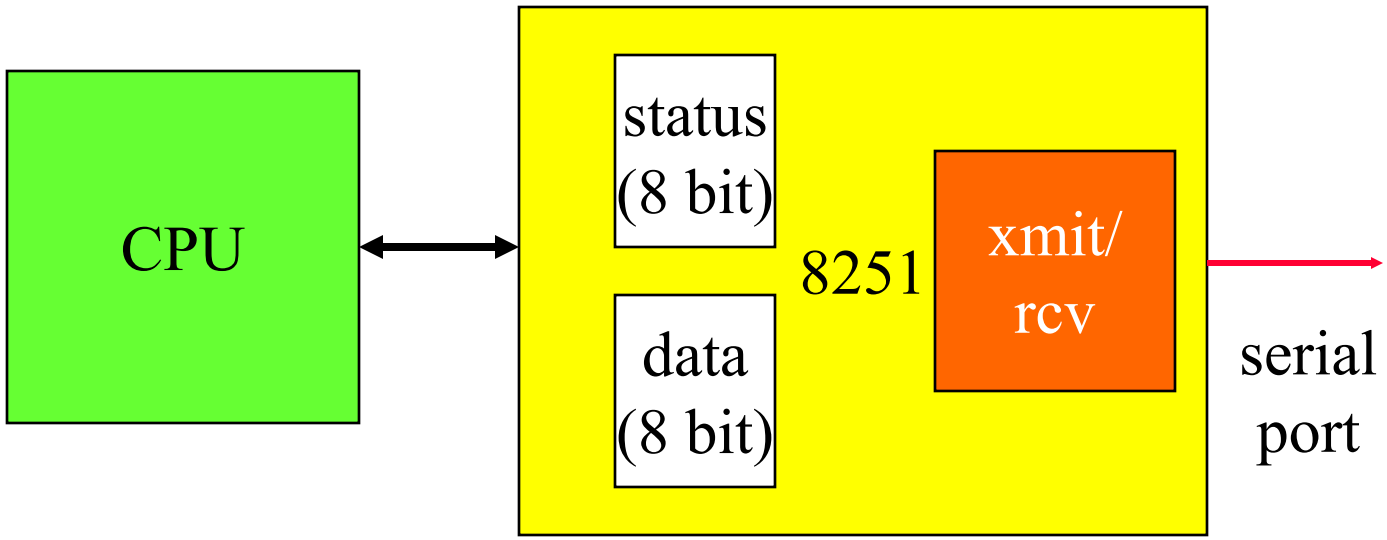
- Characters are transmitted separately:



Serial communication parameters

- Baud (bit) rate.
- Number of bits per character.
- 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:

```
DEV1 EQU 0x1000
```

- Read/write code:

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

SHARC memory mapped I/O

- Device must be in external memory space (above 0x400000).
- Use DM to control access:

```
I0 = 0x400000;
M0 = 0;
R1 = DM(I0,M0);
```

Peek and poke

- Traditional HLL interfaces:

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

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

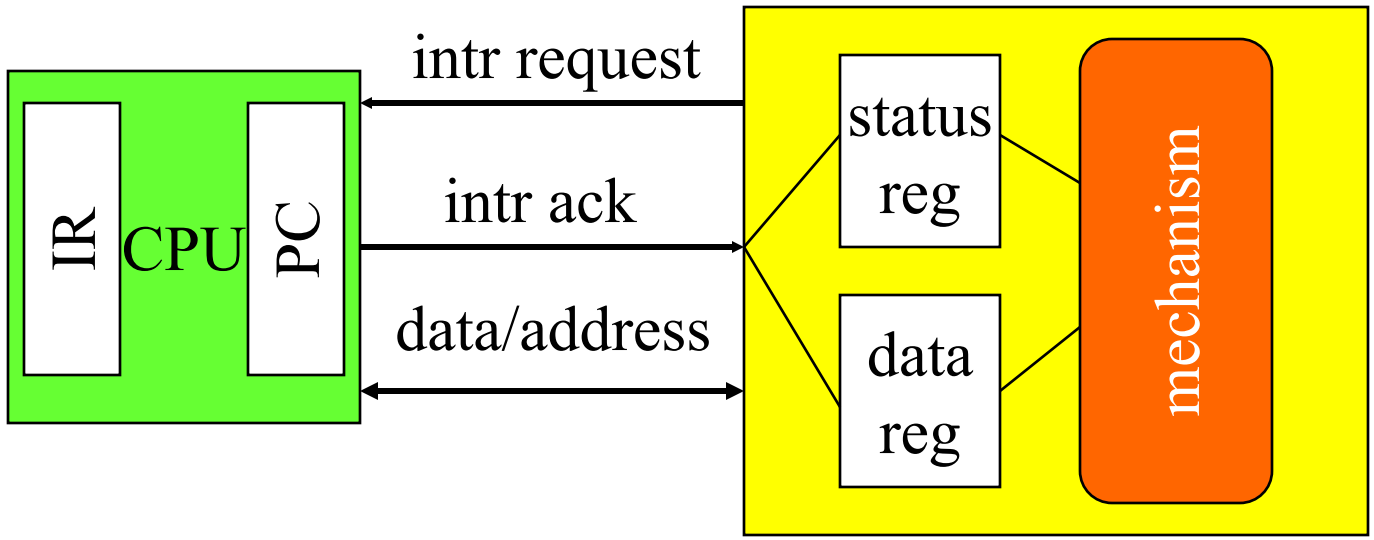
Simultaneous busy/wait input and output

```
while (TRUE) {
  /* read */
  while (peek(IN_STATUS) == 0);
  achar = (char)peek(IN_DATA);
  /* write */
  poke(OUT_DATA,achar);
  poke(OUT_STATUS,1);
  while (peek(OUT_STATUS) != 0);
  }
```

Interrupt I/O

- Busy/wait 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.
- Interrupt forces next instruction to be a subroutine call to a predetermined location.
  - Return address is saved to resume executing foreground program.

Interrupt physical interface

- CPU and device are connected by CPU bus.
- CPU and device handshake:
  - device asserts interrupt request;
  - CPU asserts interrupt acknowledge when it can handle the interrupt.

Example: character I/O handlers

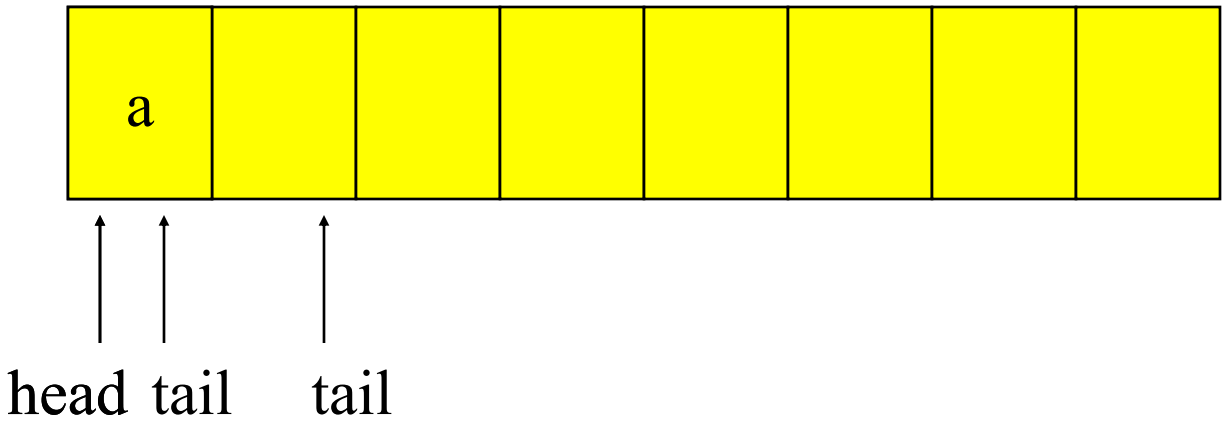
```
void input_handler() {
  achar = peek(IN_DATA);
  gotchar = TRUE;
  poke(IN_STATUS,0);
}
void output_handler() {
}
```

Example: interrupt-driven main program

```
main() {
  while (TRUE) {
     if (gotchar) {
          poke(OUT_DATA,achar);
          poke(OUT_STATUS,1);
          gotchar = FALSE;
          }
     }
  }
```

Example: interrupt I/O with buffers

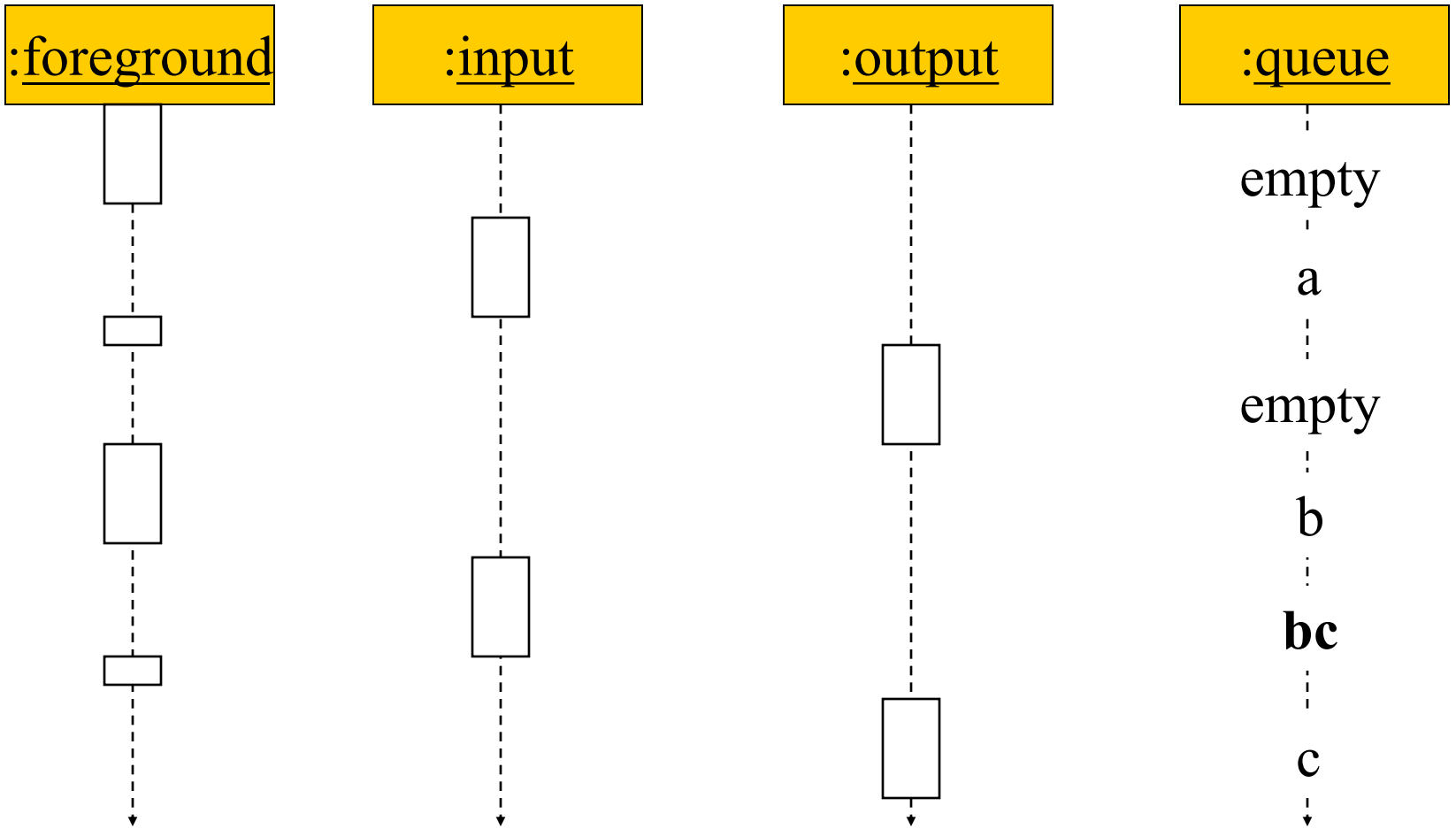
- Queue for characters:



Buffer-based input handler

```
void input_handler() {
  char achar;
  if (full_buffer()) error = 1;
  else { achar = peek(IN_DATA);
  add_char(achar); }
  poke(IN_STATUS,0);
  if (nchars == 1)
     { poke(OUT_DATA,remove_char();
  poke(OUT_STATUS,1); }
  }
```

I/O sequence diagram



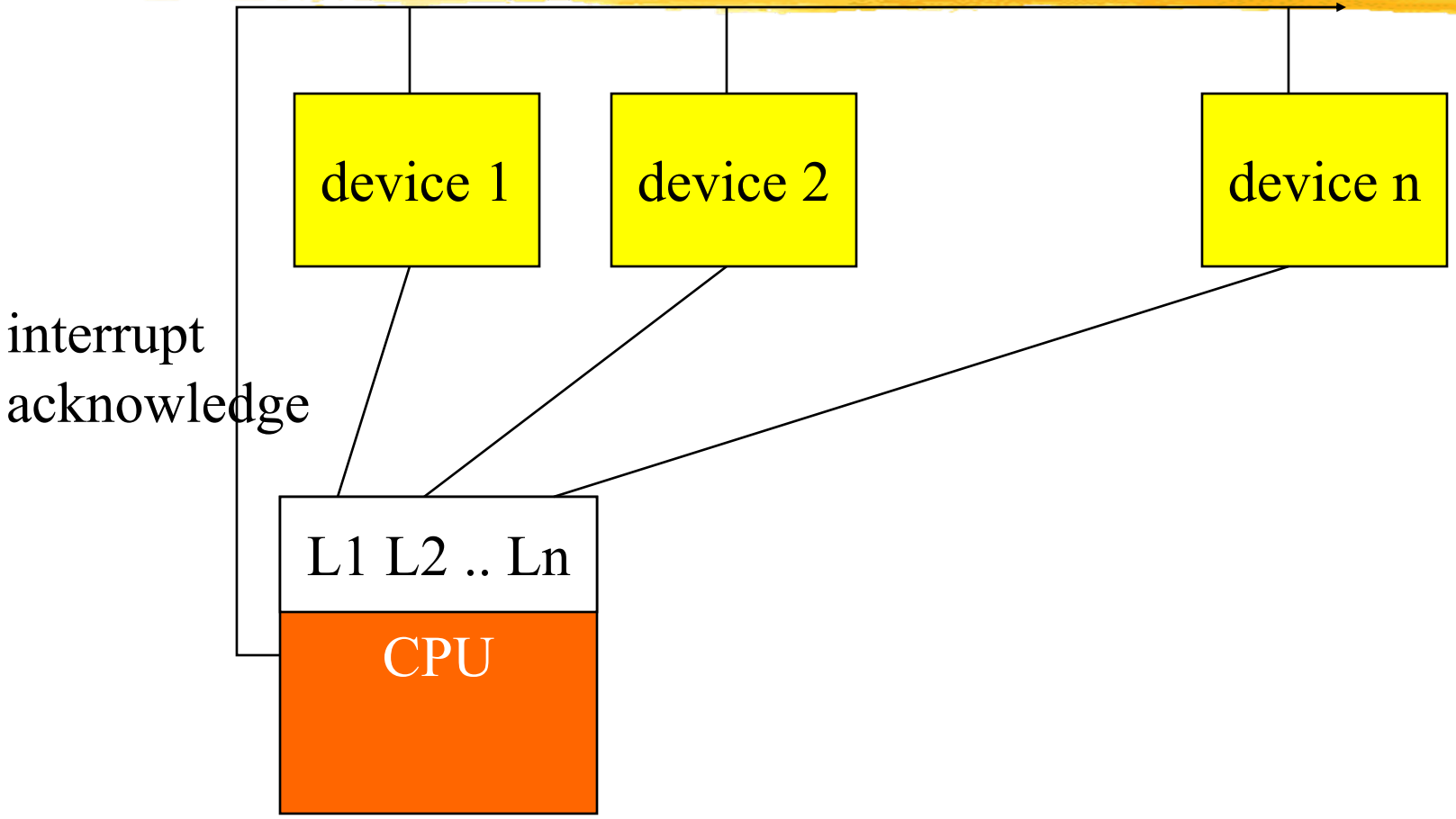
Debugging interrupt code

- What if you forget to change registers?
  - Foreground program can exhibit mysterious bugs.
  - Bugs will be hard to repeat---depend on interrupt timing.

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.

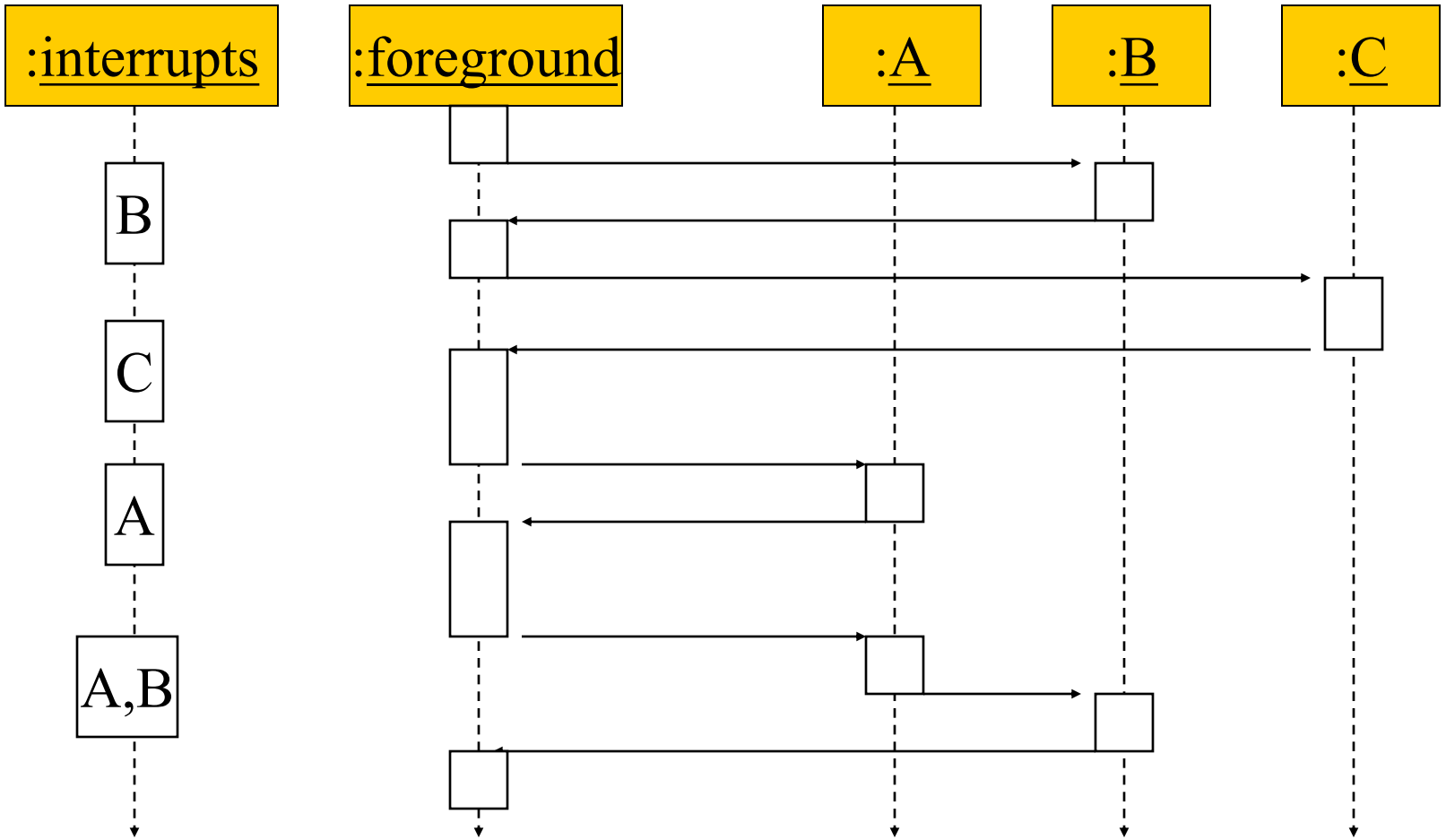
Prioritized interrupts



Interrupt prioritization

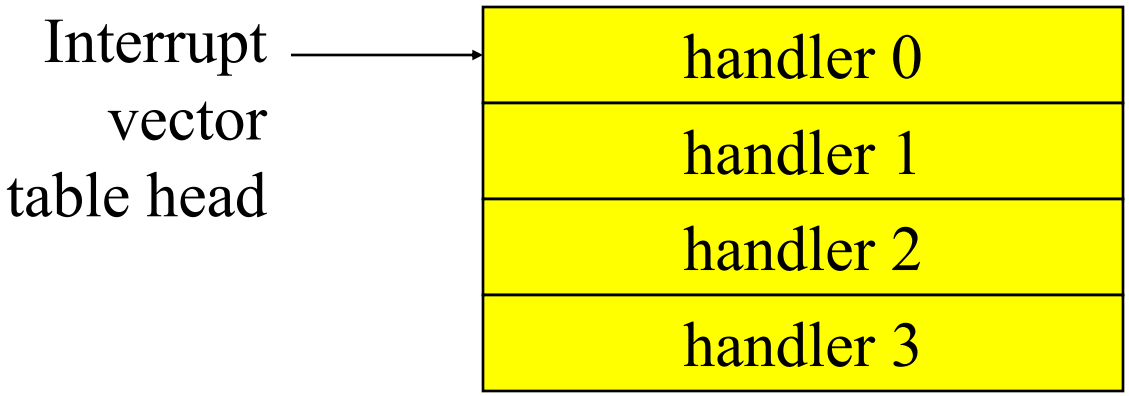
- Masking: interrupt with priority lower than current priority is not recognized until pending interrupt is complete.
- Non-maskable interrupt (NMI): highest-priority, never masked.
  - Often used for power-down.

Example: Prioritized I/O

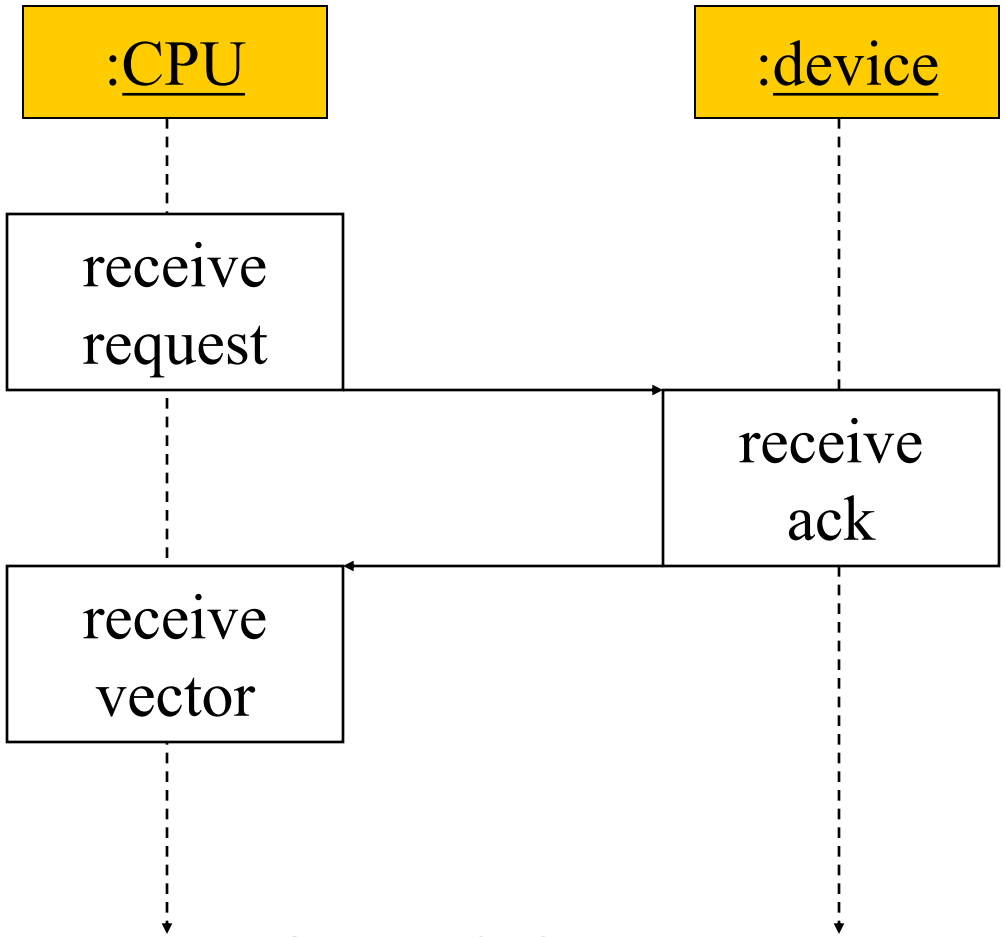


Interrupt vectors

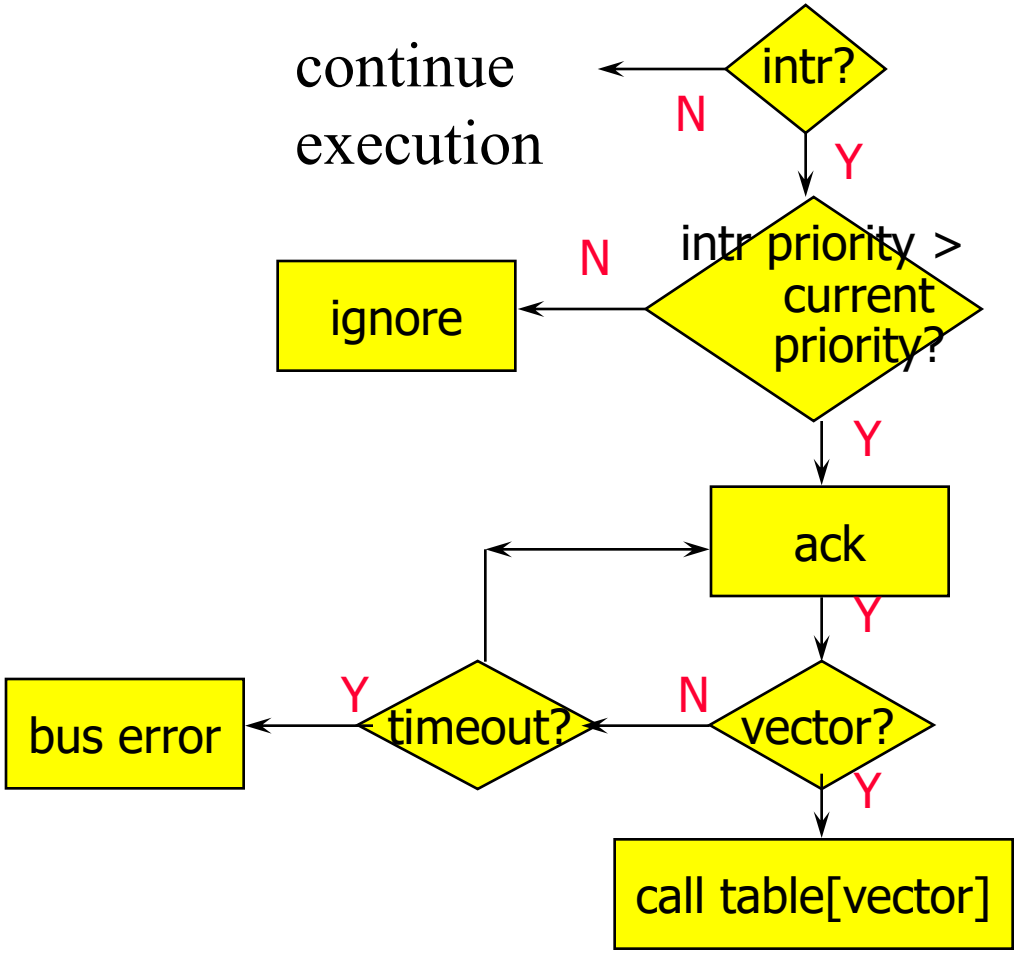
- Allow different devices to be handled by different code.
- Interrupt vector table:



Interrupt vector acquisition



Generic interrupt mechanism



Assume priority selection is handled before this point.

Interrupt sequence

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

Sources of interrupt overhead

- Handler execution time.
- Interrupt mechanism overhead.
- Register save/restore.
- Pipeline-related penalties.
- Cache-related penalties.

ARM interrupts

- ARM7 supports two types of interrupts:
  - Fast interrupt requests (FIQs).
  - Interrupt requests (IRQs).
- Interrupt table starts at location 0.

ARM interrupt procedure

- CPU actions:
  - Save PC. Copy CPSR to SPSR.
  - Force bits in CPSR to record interrupt.
  - Force PC to vector.
- Handler responsibilities:
  - Restore proper PC.
  - Restore CPSR from SPSR.
  - Clear interrupt disable flags.

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.

SHARC interrupt structure

- Interrupts are vectored and prioritized.
- Priorities are fixed: reset highest, user SW interrupt 3 lowest.
- Vectors are also fixed. Vector is offset in vector table. Table starts at 0x20000 in internal memory, 0x40000 in external memory.v

SHARC interrupt sequence

Start: must be executing or IDLE/IDLE16.

1. Output appropriate interrupt vector address.
2. Push PC value onto PC stack.
3. Set bit in interrupt latch register.
4. Set IMASKP to current nesting state.

SHARC interrupt return

Initiated by RTI instruction.

1. Return to address at top of PC stack.
2. Pop PC stack.
3. Pop status stack if appropriate.
4. Clear bits in interrupt latch register and IMASKP.

SHARC interrupt performance

Three stages of response:

- 1 cycle: synchronization and latching;
- 1 cycle: recognition;
- 2 cycles: brancing to vector.

Total latency: 3 cycles.

Multiprocessor vector interrupts have 6 cycle latency.

Supervisor mode

- May want to provide protective barriers between programs.
  - Avoid memory corruption.
- Need supervisor mode to manage the various programs.
- SHARC does not have a supervisor mode.

ARM supervisor mode

- Use SWI instruction to enter supervisor mode, similar to subroutine:

```
SWI CODE_1
```

- Sets PC to 0x08.
- Argument to SWI is passed to supervisor mode code.
- Saves CPSR in SPSR.

Exception

- Exception: internally detected error.
- Exceptions are synchronous with instructions but unpredictable.
- Build exception mechanism on top of interrupt mechanism.
- Exceptions are usually prioritized and vectorized.

Trap

- Trap (software interrupt): an exception generated by an instruction.
  - Call supervisor mode.
- ARM uses SWI instruction for traps.
- SHARC offers three levels of software interrupts.
  - Called by setting bits in IRPTL register.

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 and 2.