

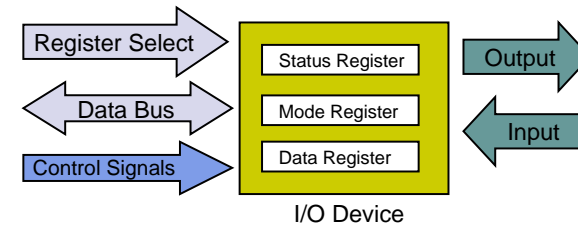
ARM CPU Input/Output, Exceptions

Ingo Sander
ingo@imit.kth.se



Input and Output Devices

- Input/Output Devices are used to communicate with the environment
- An example is a UART (Universal Asynchronous Receiver/Transmitter)
- These devices (like other peripheral devices) are controlled by reading and writing to registers



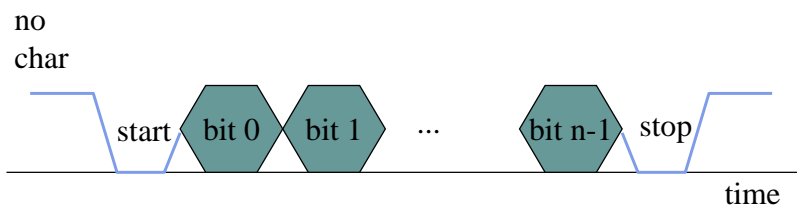
September 3, 2004

2B1447 Embedded Systems

2

Serial communication

- Characters are transmitted separately



© 2000 Morgan Kaufman
(Wayne Wolf)

September 3, 2004

2B1447 Embedded Systems

3

Universal Asynchronous Receiver/Transmitter (UART)

- Component for serial to parallel conversion
- Has a serial receiver/transmitter
- Many parameters can be configured
 - Baud rate
 - Number of bits per character
 - Parity bits
 - Length of Stop Bit

September 3, 2004

2B1447 Embedded Systems

4



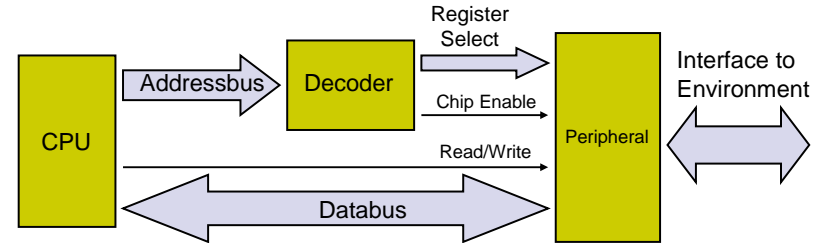
Memory-Mapped I/O

- Peripheral Components can be connected to the processor by memory-mapped I/O
- The components can be reached via a separate address space
- Memory-mapped I/O requires extra hardware for address decoding



Memory-Mapped I/O

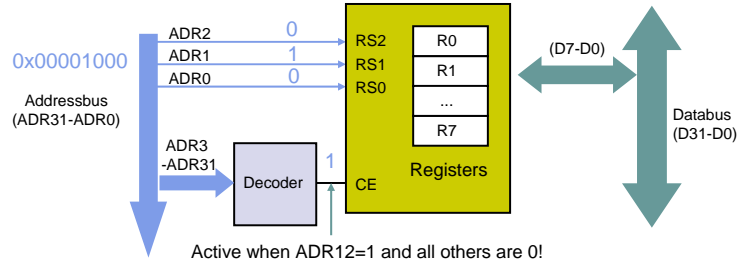
- The output chip-enable has to be active, when the input of the decoder is a correct address
- Other address bits are used for register select
- The decoder can be implemented with a small block of programmable logic or custom hardware (VHDL)



Example Memory-Mapped I/O



- A device with 8 8-bit-registers shall be connected to the address 0x1000



- The registers can now be accessed in the address space 0x1000 (R0) until 0x1007 (R7)

```
LDR r1, =0x1002
MOV r3, #7f
STRB r3, [r1]
```

- stores the lowest bytes of processorregister r3 in deviceregister R2



Accessing Memory Locations in C

- Symbolic names can be defined for memory locations

```
#define MEM_LOCATION 0x18
```

- Functions can be defined to access memory

```
• peek can be used to read a memory location (byte)
char peek(char *location)
{
    return *location;
}

• poke can be used to write to a memory location (byte)
void poke(char *location, char newval)
{
    *location = newval;
}
```



Busy Wait I/O

- Busy Wait I/O is the most basic way to communicate with an I/O-device
- The processor wait until the I/O-device has completed its current task
- Disadvantage: Processor cannot be used for other tasks during the waiting period!
- This method is also often called polling!

Example: Sending string via serial link

```

• Busy Wait I/O Pseudo Code:

Characters = String;

While not all characters sent
  Send next character;
  While Sender = Busy
    Wait;

Done!

```

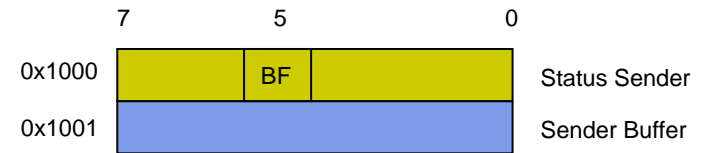


C-Programming Testing of Bits

- In order to test specific bits, it is needed to mask the other bits

Example:

- Busy Flag: Busy = 1; Non-Busy = 0



C-Programming Testing of Bits

```

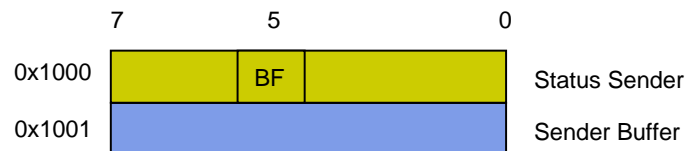
define Status 0x1000
define SendBuf 0x1001

```

```

char *myString = "Hello World";
char *current_char;

```

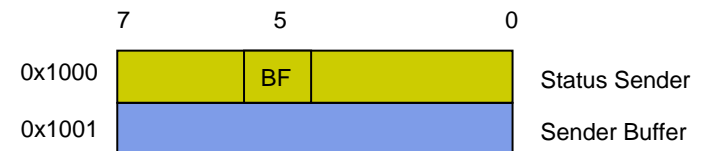


C-Programming Testing of Bits

```

While current_char != '\0';
  poke(SendBuf, *current_char);
  while ((peek(Status) & 0x20) != 0)
    ;
  /* Mask needed, since other bits      */
  /* in status register may not be zero */

```



Simultaneous busy/wait input and output



Example: Copying Characters from Input to Output

- Busy Wait I/O Pseudo Code:

```
Loop
  While inBuffer busy
    Wait;
  Read Character
  Copy Character to Output Buffer
  Send Character
  While outBuffer busy
    Wait;
```

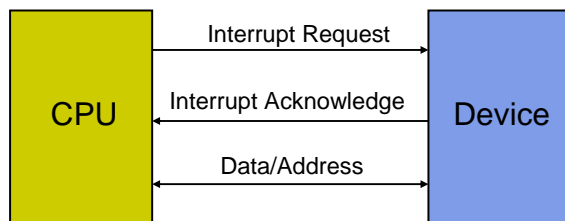
Interrupt I/O



- 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.

© 2000 Wolf (Morgan Kaufman)

Interrupt Scheme



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.

© 2000 Wolf (Morgan Kaufman)



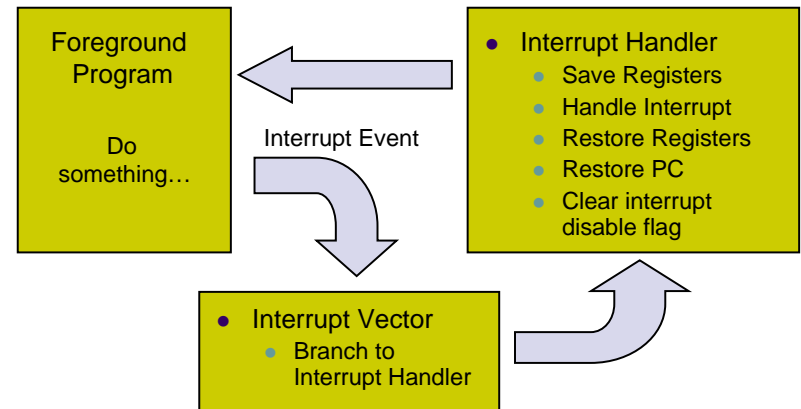
Interrupt behavior

- 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*

© 2000 Wolf (Morgan Kaufman)



Programming Interrupt



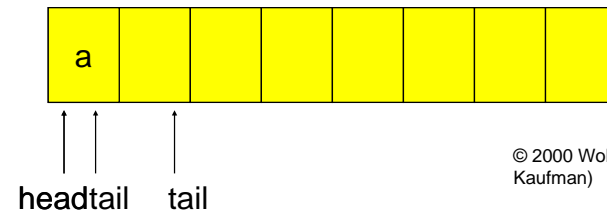
ARM interrupt procedure

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



Example: interrupt I/O with buffers (Wolf)

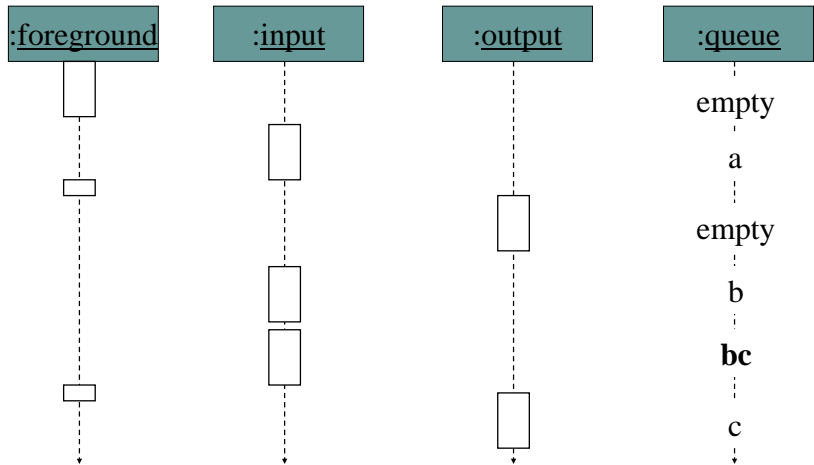
- Independent receive, send realized by two interrupt routines
 - Receive-interrupt routine
 - Puts a character into queue
 - Send-interrupt routine
 - Sends a character, when sender ready



© 2000 Wolf (Morgan Kaufman)



I/O sequence diagram (Wolf)



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
 - It is difficult to debug an interrupt routine!

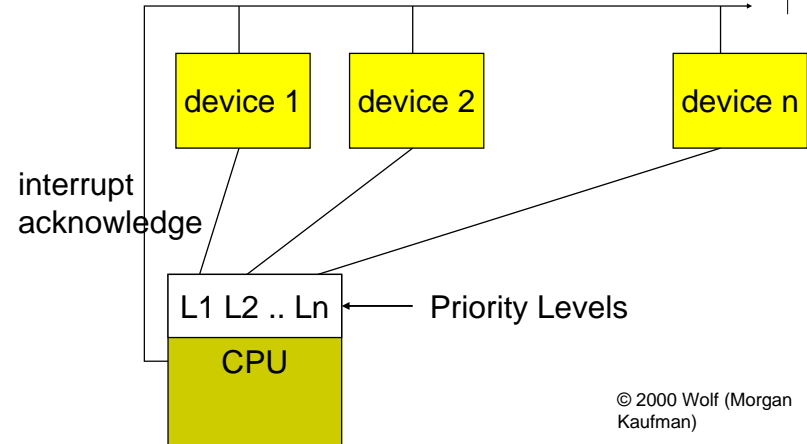


Priorities and vectors

- 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





Prioritized Interrupts

- Some CPUs support several interrupt levels by their hardware
- Otherwise extra hardware (priority decoder) can be used to create several levels of interrupt

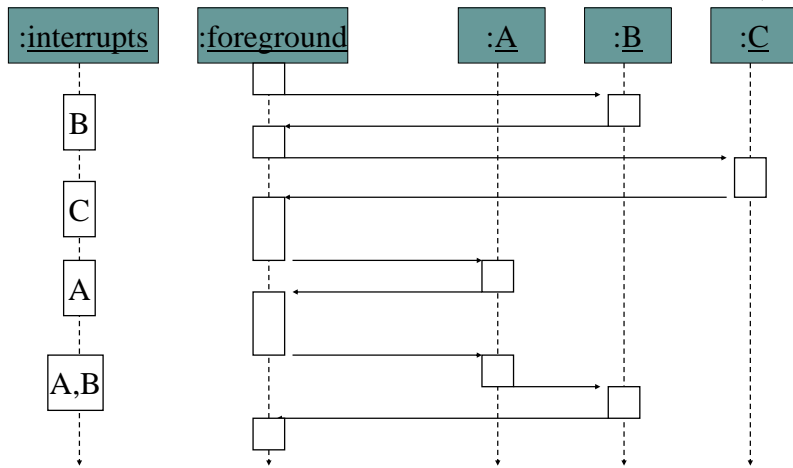


Interrupt prioritization

- **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.

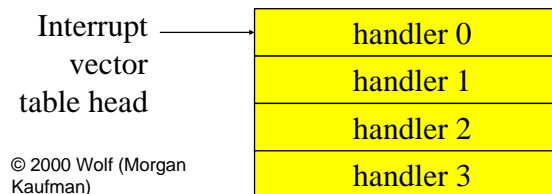


Example: Prioritized I/O



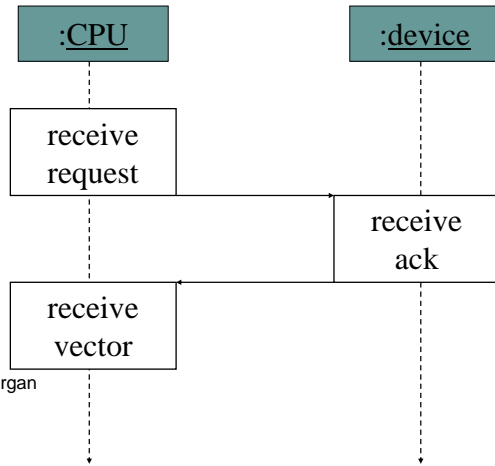
Interrupt vectors

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





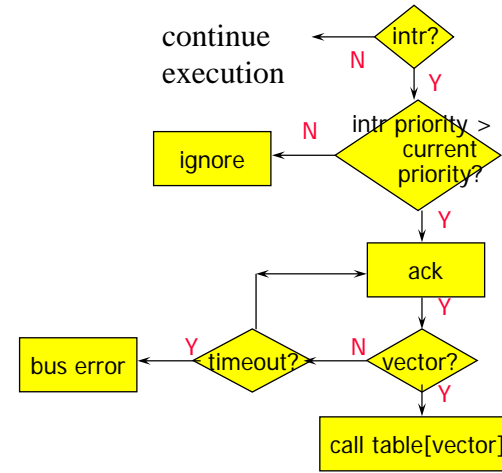
Interrupt vector acquisition



© 2000 Wolf (Morgan Kaufman)



Generic interrupt mechanism



Assume priority selection is handled before this point.

© 2000 Wolf (Morgan Kaufman)



Interrupt sequence

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

© 2000 Wolf (Morgan Kaufman)



Sources of interrupt overhead

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

© 2000 Wolf (Morgan Kaufman)



ARM interrupts

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

© 2000 Wolf (Morgan Kaufman)



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

© 2000 Wolf (Morgan Kaufman)



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

© 2000 Wolf (Morgan Kaufman)

- *Other sources treat an interrupt also as exception... (Ingo Sander)*



Exceptions

- *Reset*: at start up
- *Undefined Instruction*: when the processor cannot decode an instruction
- *Software Interrupt*: when an SWI instruction is executed (used for operating system routines)
- *Prefetch Abort*: when the processor attempts to fetch an instruction from an address without the correct access permissions
- *Data Abort*: when the processor attempts to fetch data from an address without the correct access permissions
- *Interrupt Request*: used by external hardware to interrupt the normal execution (IRQ and FIQ)



Trap

- *Trap (software interrupt):* an exception generated by an instruction.
 - Call supervisor mode
- ARM uses SWI instruction for traps
SWI SYS_CALL
- The interrupt handler has to analyse the instruction in order to find out which system call has been issued!
 - SWI-Format: SWI immediate



Trap Example for SWI Handler

```

SWI_Handler
    ; Store Registers
    STMFD sp!, {r0-r12, lr}
    ; Read SWI Instruction
    LDR r10, [lr, #-4]
    ; Mask off top 8 bits
    BIC r10, r10, #0xff000000
    ; SWI Number is stored in lower 24 bits
    ; r10 - contains the SWI handler
    BL handler_routine
    ; Restore Registers and PC
    LDMFD sp!, {r0-r12, pc}^ ← refers to user mode registers
  
```

Exeptions Mode, Priority, Vector Table



Exception	Mode	Priority	Address
Reset	SVC	1	0x00000000
Undefined Instruction	UND	6	0x00000004
Software Interrupt	SVC	6	0x00000008
Prefetch Abort	ABT	5	0x0000000C
Data Abort	ABT	2	0x00000010
Reserved	-	-	-
Interrupt Request	IRQ	4	0x00000018
Fast Interrupt Request	FIQ	3	0x0000001C

Exeptions Link Register Offsets



Exception	Address	Use
Reset	-	Lr is not defined for Reset
Undefined Instruction	lr	Points to the next instruction after the undefined instruction
Software Interrupt	lr	Points to the next instruction after the SWI instruction
Prefetch Abort	lr-4	Points to instruction that caused the prefetch abort exeption
Data Abort	lr-8	Points to instruction that caused the data abort exeption
Reserved	-	-
Interrupt Request	lr-4	Return address from IRQ Handler
Fast Interrupt Request	lr-4	Return address from FIQ Handler



Interrupt Return Address

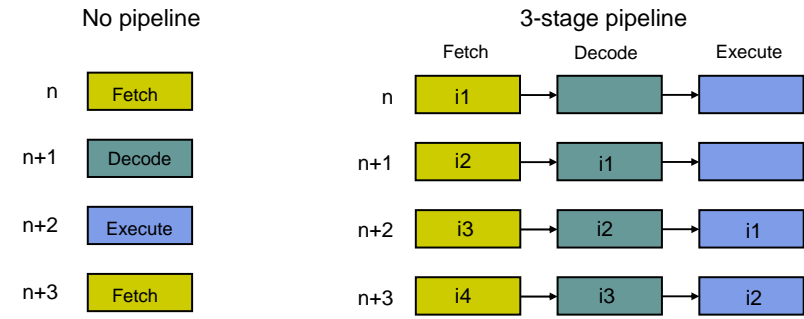
Why "Ir - 4"?

- When an interrupt occurs the link register points to last executed instruction
 - $Ir = pc + 8$ (since PC points to fetched instruction)
- Thus the return address is
 - $pc + 4 = Ir - 4$



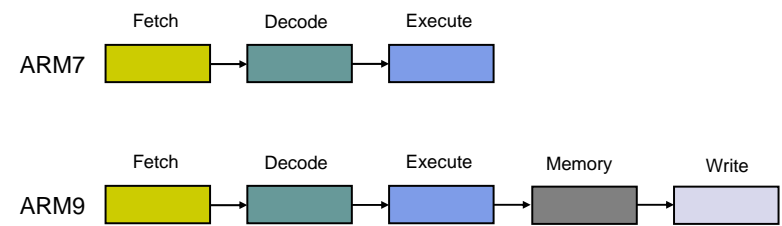
Pipelines

- A pipeline is a mechanism to increase the number of executed instructions per cycle



ARM7 and ARM9 pipeline

- The ARM7 pipeline has 3 stages
- The ARM9 pipeline has 5 stages



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.



Summary

- Interrupts are very suitable for I/O
- They work in a similar way as subroutines, but are initiated by external events
- Debugging of interrupt routines is difficult
- Registers should be stored on stack before being modified in interrupt routine
- There are other exceptions that work in the same way as interrupt