

CHAPTER 8

Programming in Assembly Part 4: I/O Programming

Relative Speed

- I/O devices are sometimes mechanical devices (e.g., solenoids, relays, etc.) that take a long time to perform an action.
- The computer performs operations orders of magnitude faster than the I/O devices.
- **Synchronization**: The CPU must wait for the I/O device to finish each command before issuing the next.

Asynchronous Events

- The events that determine when an input device has data available or when an output device needs data are **independent** of the CPU.
- I/O programming, therefore, requires "**hand-shaking**" between the CPU and the I/O device to coordinate the transfer so that data is transferred reliably.

Time Behavior

<i>Data Rate</i>	<i>Pattern</i>	<i>Increasing Time →</i>
Low	Random	
	Periodic	
High	Random	
	Periodic	

I/O Data Ports

Output Device: Wait for status port to indicate that device is ready to accept data, then copy data to output data port.

- Example: UART Transmit Holding Register

Input Device: Wait for status port to indicate that data is available, then copy data from the input data port.

- Example: UART Receive Buffer Register

I/O Status Ports

(Example: UART Line Status Register)

- **Output Device:** Status port contains a single bit that indicates whether or not the device is ready to accept new data from the data port.

7	6	5	4	3	2	1	0
Receiver Error	Trans. Empty	THRE	Break	Framing Error	Parity Error	Overrun Error	Data Ready

THRE=1 when device is ready to receive new data; writing new data into the output data port resets this bit to 0.

- **Input Device:** Status port contains a single bit that indicates when new data is available to be read from the data port.

7	6	5	4	3	2	1	0
Receiver Error	Trans. Empty	THRE	Break	Framing Error	Parity Error	Overrun Error	Data Ready

Data Ready will go to 1 when new data is available in the input data port; reading that data from the data port resets this bit to 0.

I/O Control Ports

(Example: UART Line Control Register)

- Used to control operational features of the device.

7	6	5	4	3	2	1	0
DLAB	Set Break	Stick Parity	Even Parity Select	Parity Enable	Number of Stop Bits	Word Length Select Bits	

Accessing I/O Ports

I/O Mapped:
(Desktop Processors)

Processor has an I/O address bus separate from the memory address bus used by special instructions for transferring data to and from an I/O port.

Memory-Mapped:
(Embedded Processors)

I/O devices and memory both share parts of the same memory address space. I/O devices are accessed like regular memory locations.

Three Strategies

- Polled Waiting Loops
- Interrupt-driven I/O
- Direct Memory Access (DMA)

Polled Waiting Loop

(Example: Output data to UART)

```
// THRE=1 if transmitter holding register empty
#define THRE    (1 << 5)

#define DATAPORT(base)    *(base + 0)
#define STATUSPORT(base) *(base + 24)

void Output(uint8_t data, uint8_t *uart_base)
{
    while (STATUSPORT(uart_base) & THRE) == 0)
    {
        // do nothing (wait for device to become
ready)
    }
    DATAPORT(uart_base) = data ;
}
```

Polled Waiting Loop

(Example: Input data from UART)

```
// DataReady = 1 if data available to be read
#define DATAREADY (1 << 0)
```

```
#define DATAPORT(base)      *(base + 0)
#define STATUSPORT(base)   *(base + 24)
```

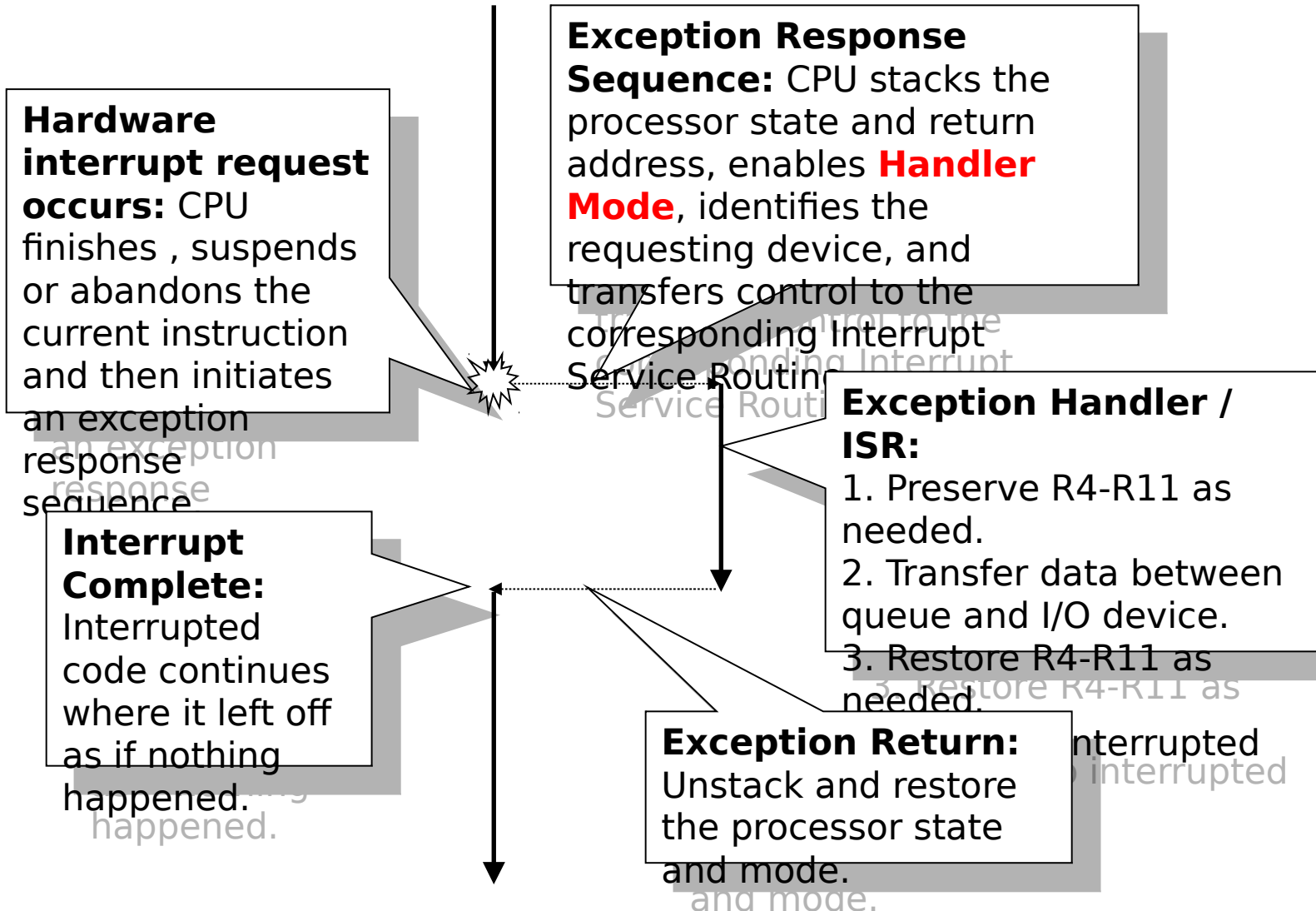
```
uint8_t Input(uint8_t *uart_base)
```

```
{
    while (STATUSPORT(uart_base) & DATAREADY) == 0)
    {
        // do nothing (wait for device to become
ready)
    }
    return DATAPORT(uart_base) ;
}
```

What if urgent data arrives on another input port while executing this

EXCEPTIONS AND INTERRUPTS

Interrupt Processing



Exceptions

Definitions:

- Any condition that needs to halt the normal sequential flow of instruction execution.
- Reset
- SVC Supervisor Call
(Software Interrupt)
- Fault
E.g., undefined opcode
- Interrupts

Exceptions

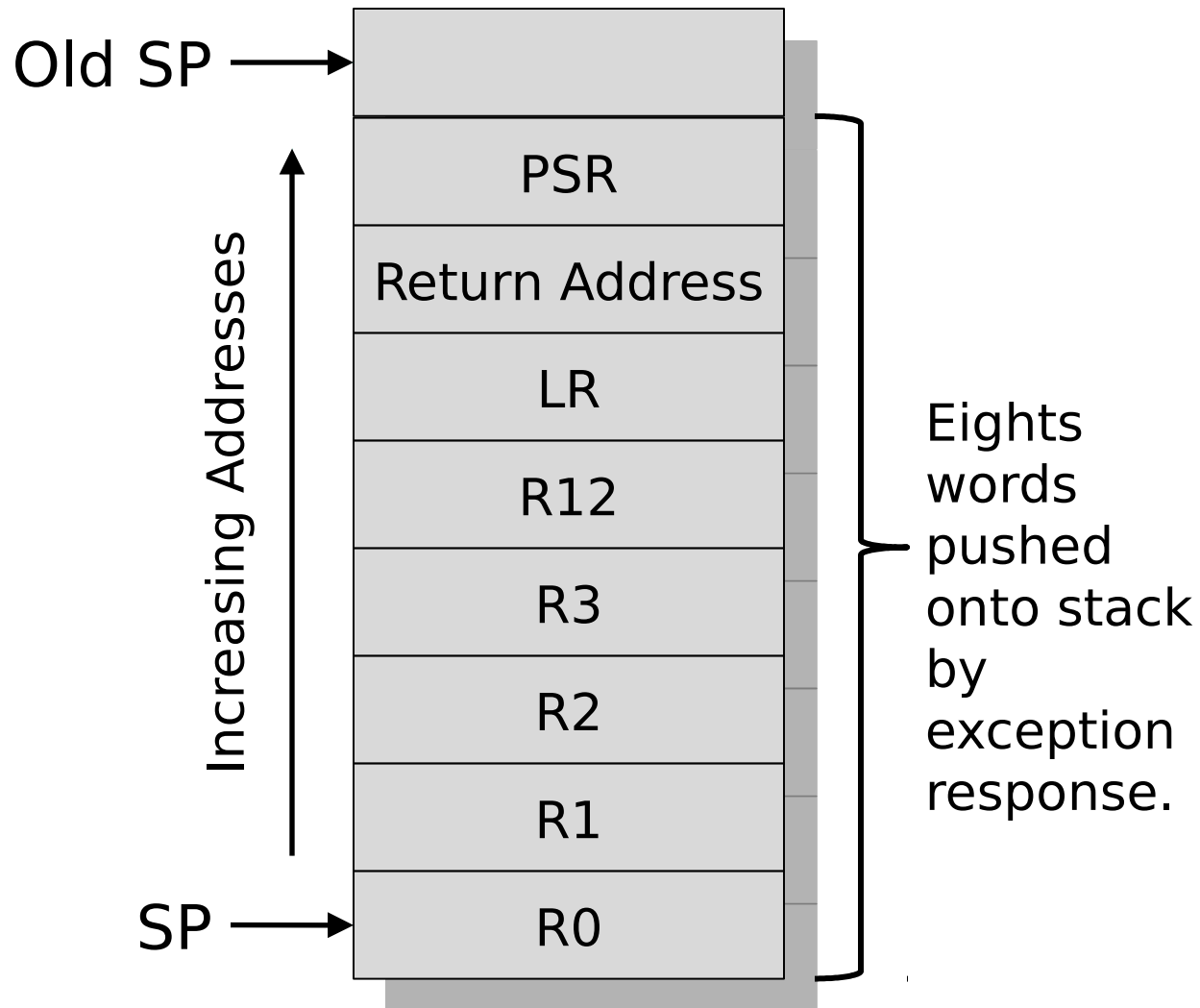
Each exception has:

1. An exception number
2. A priority level
3. An exception handler routine
4. An entry in the vector table
(Address of associated handler routine)

Exception Response

1. Processor state (8 words) stored on stack:
xPSR, Return Address, LR, R12, R3 - R0
Allows a regular C function to be an ISR!
2. Processor switched (from **Thread Mode**) to **Handler Mode**
recorded in “T” Bit (#24) of pushed xPSR.
3. PC \equiv vector table[exception #]

Interrupt Stacking



Exception Handlers

- Definition:

An exception handler is a software routine that is executed when a specific exception condition occurs.

Note: Most, but not all, exception handlers return to the previous code.

Exception Return

Exception return occurs when in *Handler Mode* and one of the following instructions is executed:

1. POP/LDM includes the PC, or
2. LDR with PC as the destination, or
3. BX with any register as the source

“Unstacks” (pops and restores) registers and return address (into PC) from the stack.

- Restores T-bit (in xPSR) and thus previous processor mode.
- Resumes suspended multi-cycle instruction (if any), or restarts abandoned instruction (if any)

LATENCY

The passage of time from the moment an event occurs until the processor executes code to process the event.

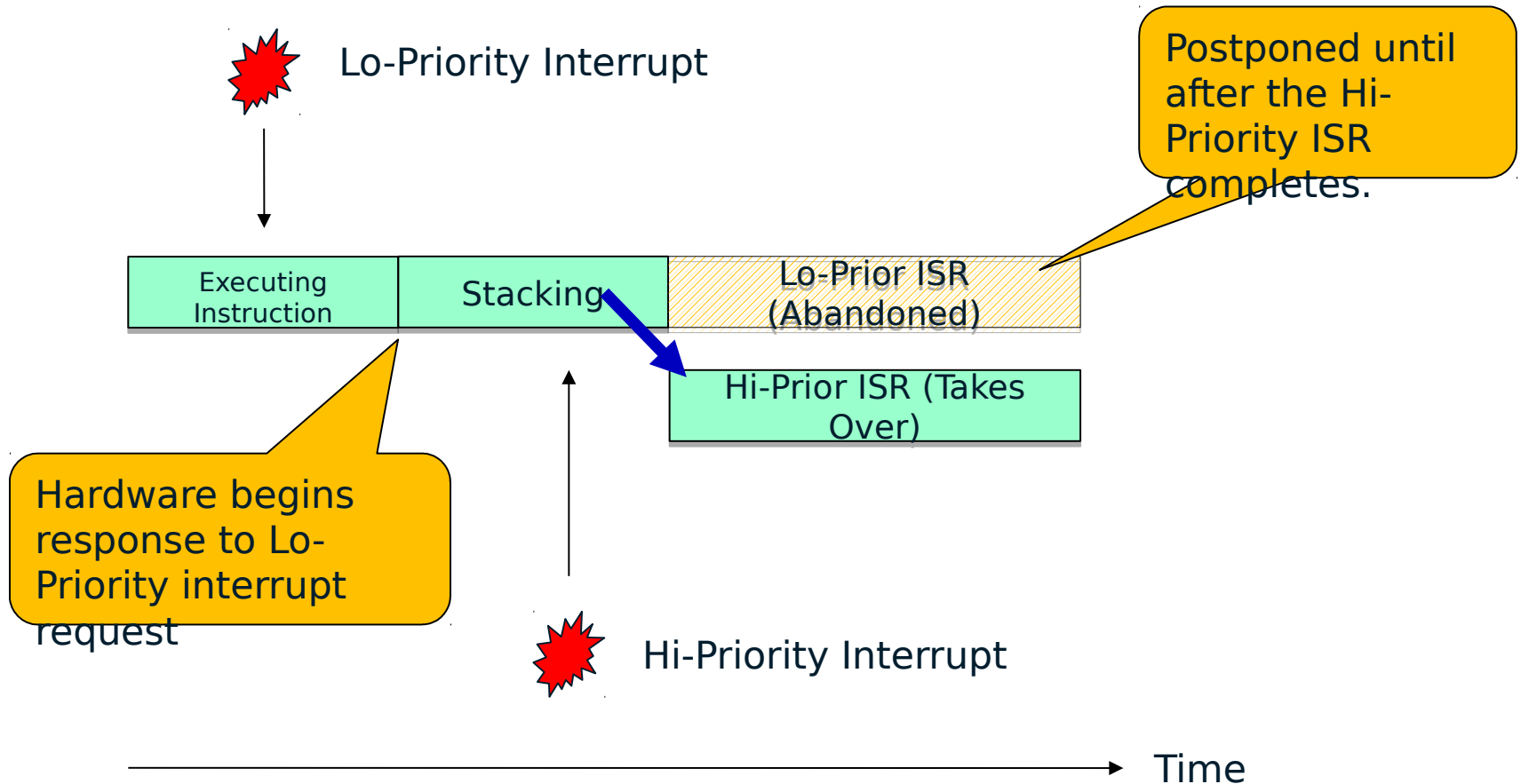
ARM Latency Reduction

1. **Abandoning or Suspending** the current instruction to start the exception response sooner.
2. **Late Arrival Processing**: Allowing a high-priority event to take over the exception response of a low-priority event.
3. **Tail-Chaining**: Eliminating unnecessary stacking and unstacking.

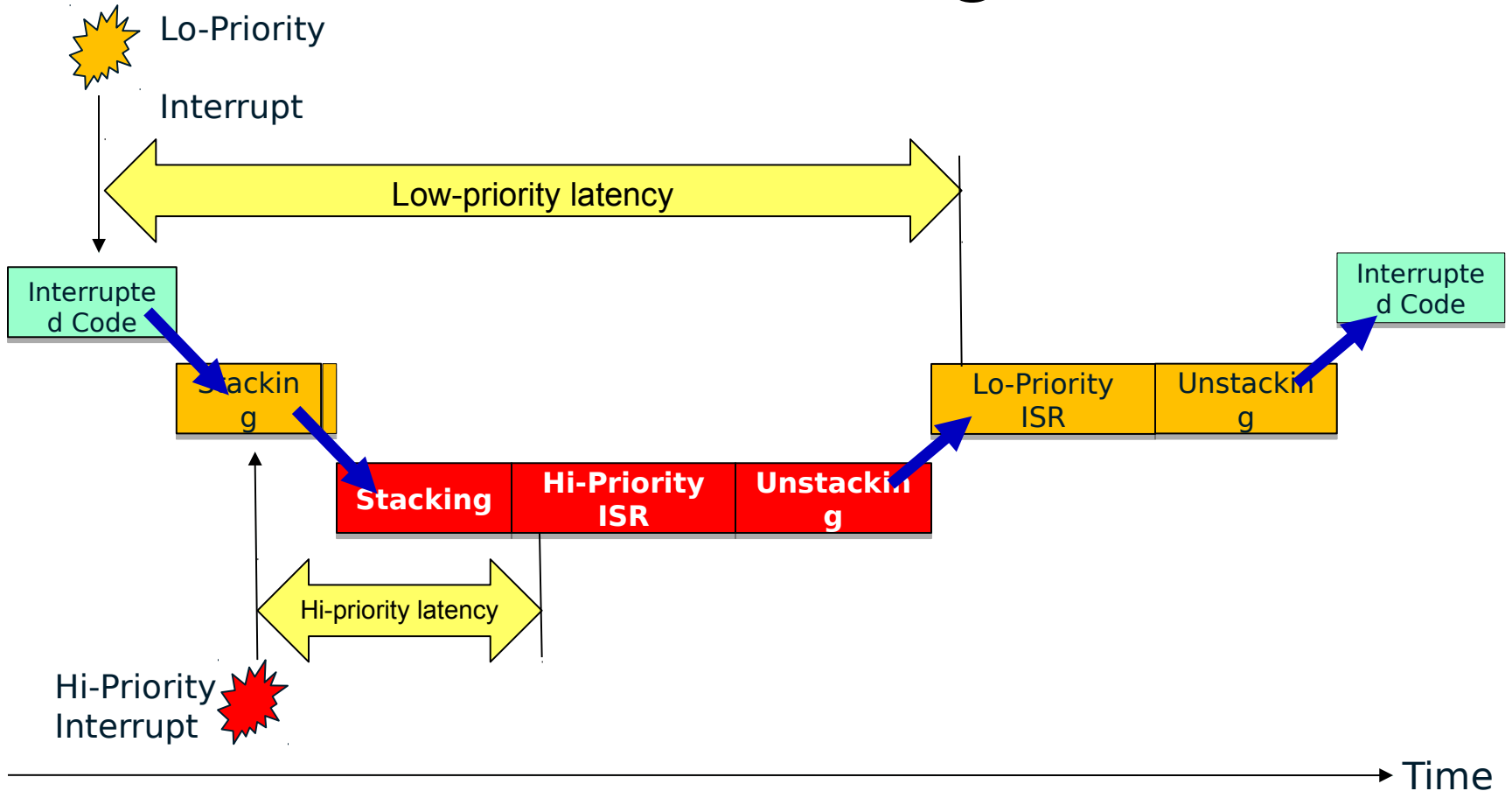
Suspending/Abandoning Instructions

1. Instructions are allowed to complete if only 1 clock cycle is required
2. Instructions that transfer multiple words to/from memory are suspended
(LDM, STM, PUSH & POP)
3. All other instructions are abandoned and restarted after the interrupt.

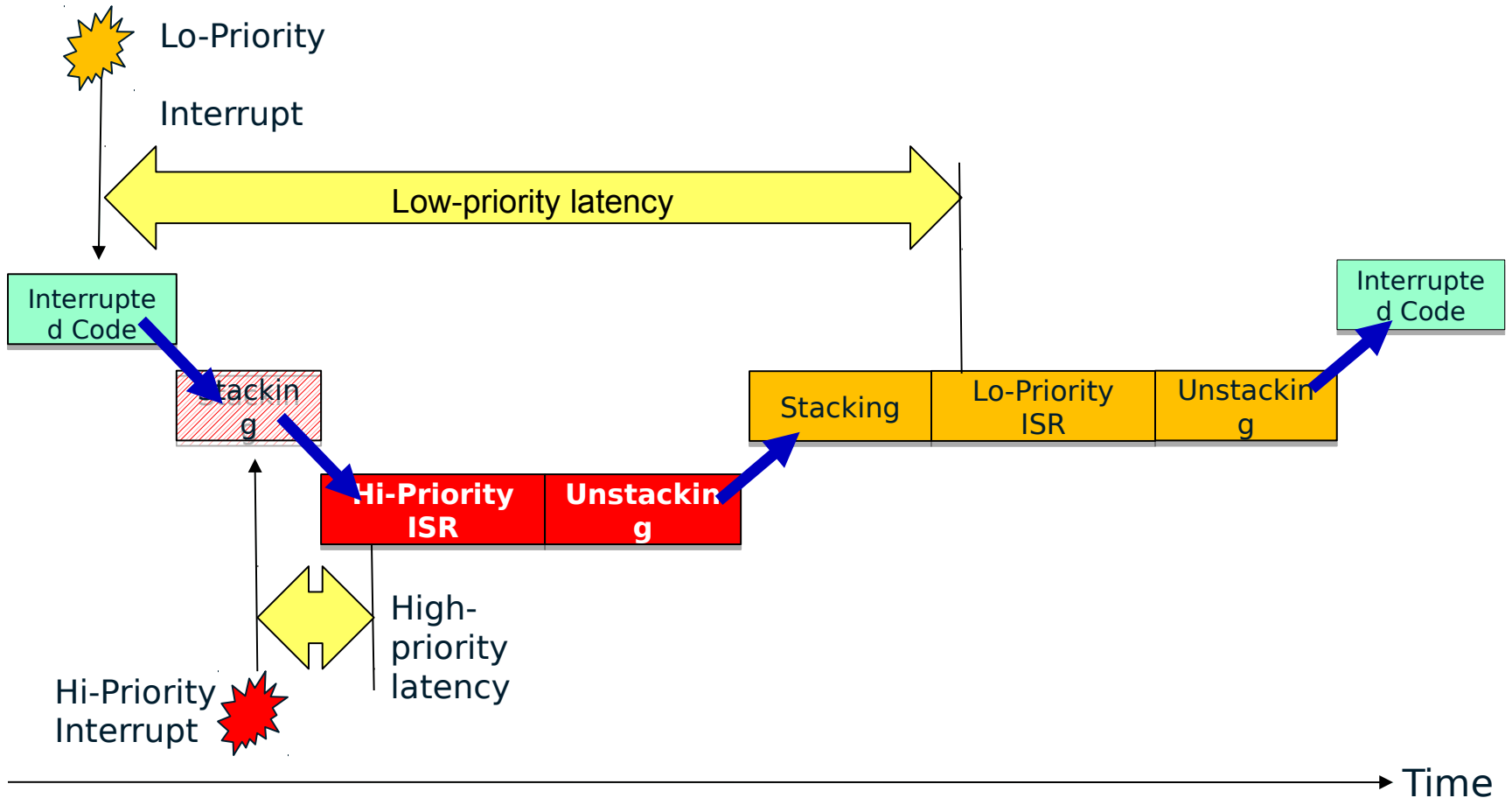
Late Arrival Processing



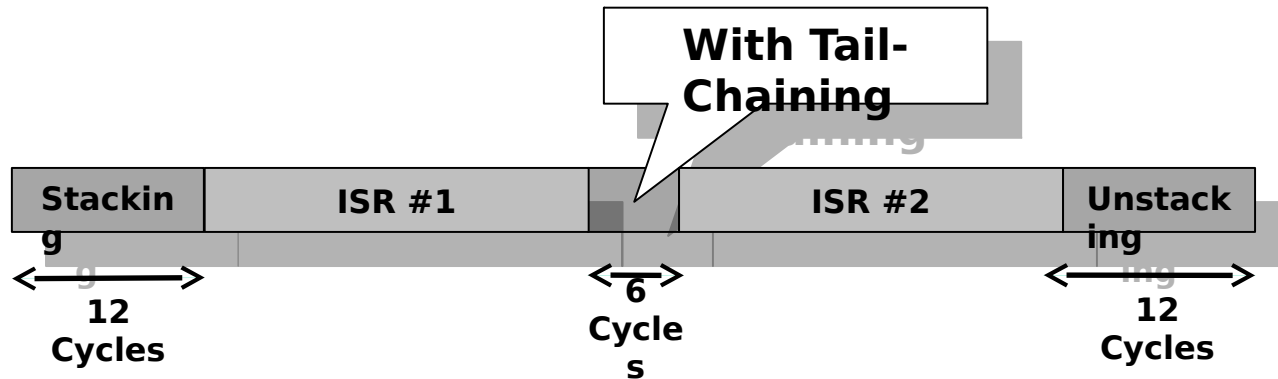
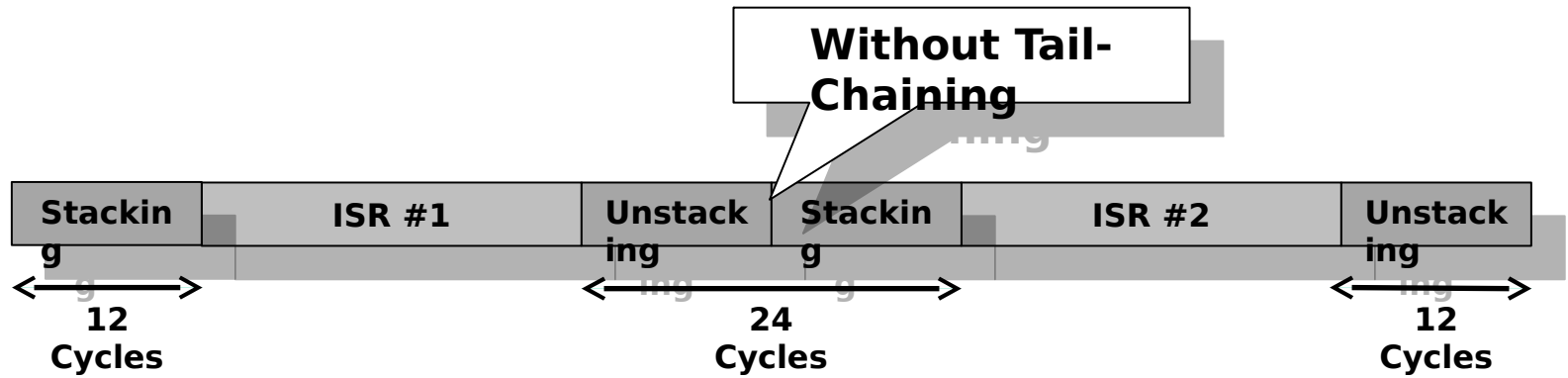
Without Late Arrival Processing



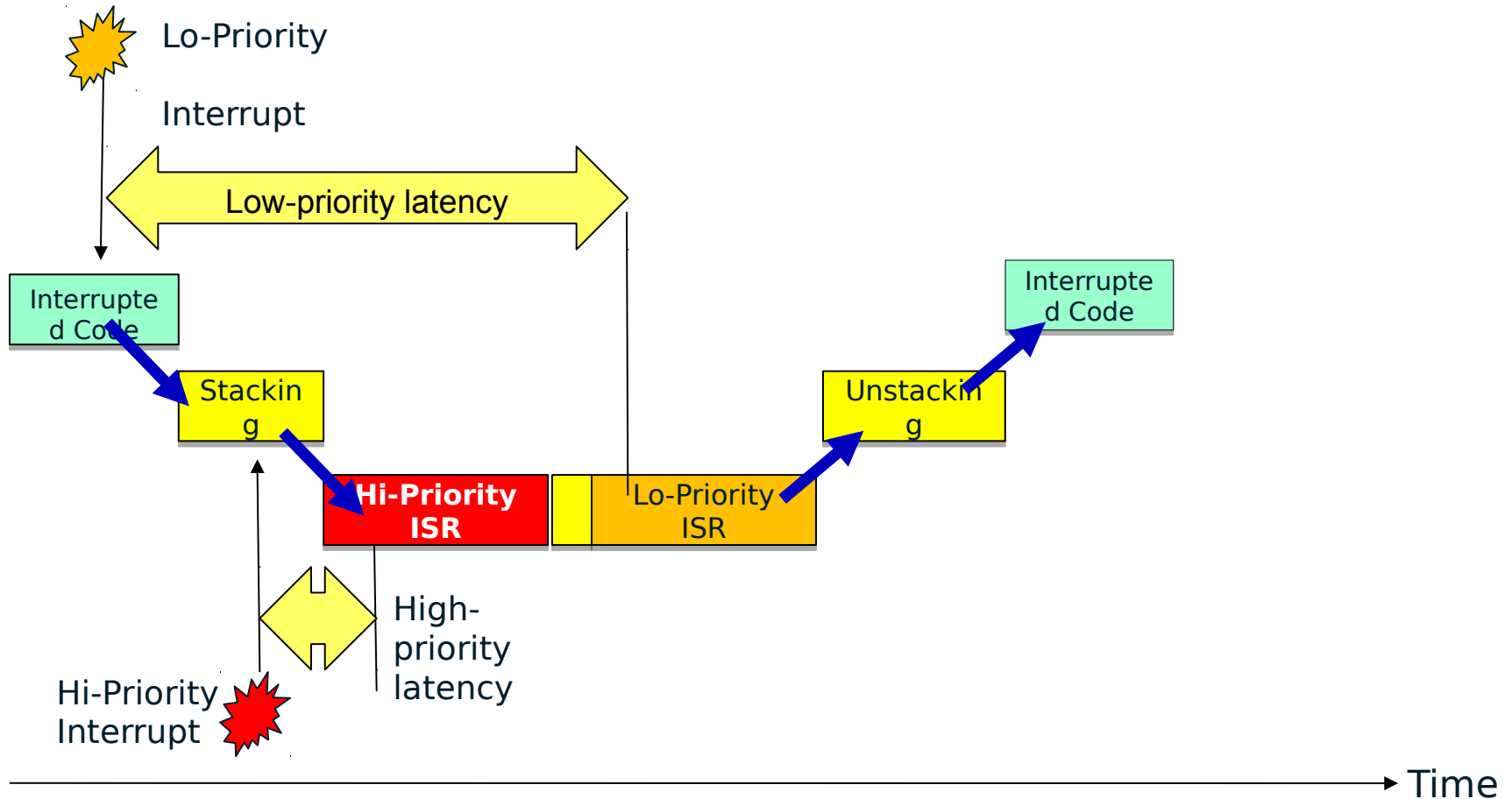
With Late Arrival Processing



Tail-Chaining



With LAP and Tail-Chaining



EXCEPTION AND INTERRUPT PRIORITIES

Nested Vectored Interrupt Controller

(Mapped to addresses E000E100-E000ECFF₁₆)

The NVIC provides the ability to:

1. Individually Enable/Disable interrupts from specific devices.
2. Establishes relative priorities among the various interrupts.

Exception Priorities and Vector Table Positions

Exception Type	Exception # ¹	IRQ #	Priority	Comment
		n/a		Initial SP value (loaded on reset)
Reset	1		-3 (fixed)	Power up and warm reset
NMI	2		-2 (fixed)	Non-Maskable Interrupt
Hard Fault	3		-1 (fixed)	
Memory Mgmt.	4		Configured through System Handler Priority Registers	
Bus Fault	5			Address/Memory-related faults
Usage Fault	6			Undefined instruction
Reserved	7-10			
SVCall	11			Software Interrupt (SVC instruction)
Debug Monitor	12			
Reserved	13			
PendSV	14			
SysTick	15			System Timer Tick
Ext. Interrupts	16-255	0-239	Configurable	

¹Vector table entries are ordered by exception number, with the first entry reserved for the initial SP value. IRQ # is the interrupt request number (same as the exception number minus 16).

Interrupt Priority Registers

- Each interrupt is assigned an 8-bit priority number:
 - Highest Priority: 0
 - Lowest Priority: 255

Enabling/Disabling Interrupts

CPSIE ; Enable **all** external
interrupts

CPSID ; Disable **all** external
interrupts

31

0

NVIC Interrupt Clear Enable Reg.

31

0

NVIC Interrupt Set Enable Reg.

Set a bit to 1 in this
register to [disable](#)
interrupts from an
external I/O device

Set a bit to 1 in this
register to [enable](#)
interrupts from an
external I/O device

NVIC External Interrupts

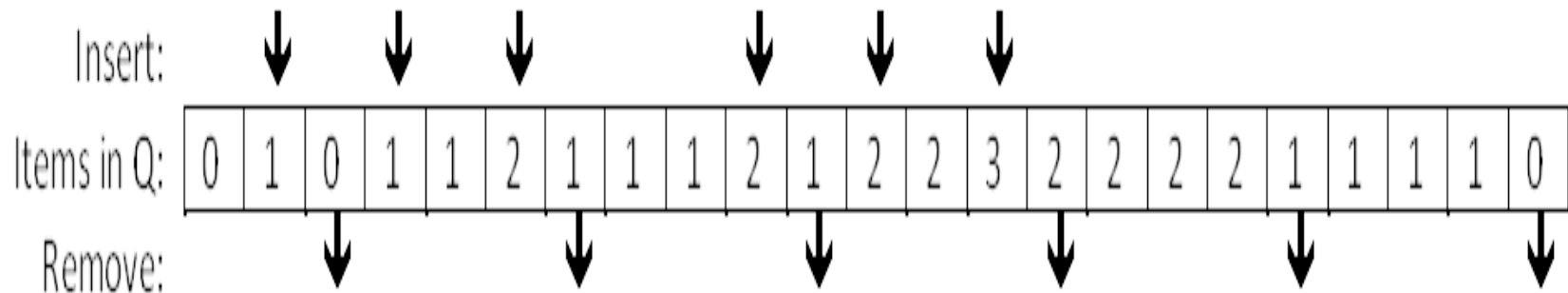
0-4	Watchdog Timer
5-24	UART0-21
25	SAI
26	SAI
27	SAI
28	SAI
29-31	SAI
32-35	SAI
36-39	SAI
40-43	SAI
44-47	SAI
48-51	SAI
52-55	SAI
56-59	SAI
60-63	SAI
64-67	SAI
68-71	SAI
72-75	SAI
76-79	SAI
80-83	SAI
84-87	SAI
88-91	SAI
92-95	SAI
96-99	SAI
100-103	SAI
104-107	SAI
108-111	SAI
112-115	SAI
116-119	SAI
120-123	SAI
124-127	SAI
128-131	SAI
132-135	SAI
136-139	SAI
140-143	SAI
144-147	SAI
148-151	SAI
152-155	SAI
156-159	SAI
160-163	SAI
164-167	SAI
168-171	SAI
172-175	SAI
176-179	SAI
180-183	SAI
184-187	SAI
188-191	SAI
192-195	SAI
196-199	SAI
200-203	SAI
204-207	SAI
208-211	SAI
212-215	SAI
216-219	SAI
220-223	SAI
224-227	SAI
228-231	SAI
232-235	SAI
236-239	SAI
240-243	SAI
244-247	SAI
248-251	SAI
252-255	SAI
256-259	SAI
260-263	SAI
264-267	SAI
268-271	SAI
272-275	SAI
276-279	SAI
280-283	SAI
284-287	SAI
288-291	SAI
292-295	SAI
296-299	SAI
300-303	SAI
304-307	SAI
308-311	SAI
312-315	SAI
316-319	SAI
320-323	SAI
324-327	SAI
328-331	SAI
332-335	SAI
336-339	SAI
340-343	SAI
344-347	SAI
348-351	SAI
352-355	SAI
356-359	SAI
360-363	SAI
364-367	SAI
368-371	SAI
372-375	SAI
376-379	SAI
380-383	SAI
384-387	SAI
388-391	SAI
392-395	SAI
396-399	SAI
400-403	SAI
404-407	SAI
408-411	SAI
412-415	SAI
416-419	SAI
420-423	SAI
424-427	SAI
428-431	SAI
432-435	SAI
436-439	SAI
440-443	SAI
444-447	SAI
448-451	SAI
452-455	SAI
456-459	SAI
460-463	SAI
464-467	SAI
468-471	SAI
472-475	SAI
476-479	SAI
480-483	SAI
484-487	SAI
488-491	SAI
492-495	SAI
496-499	SAI
500-503	SAI
504-507	SAI
508-511	SAI
512-515	SAI
516-519	SAI
520-523	SAI
524-527	SAI
528-531	SAI
532-535	SAI
536-539	SAI
540-543	SAI
544-547	SAI
548-551	SAI
552-555	SAI
556-559	SAI
560-563	SAI
564-567	SAI
568-571	SAI
572-575	SAI
576-579	SAI
580-583	SAI
584-587	SAI
588-591	SAI
592-595	SAI
596-599	SAI
600-603	SAI
604-607	SAI
608-611	SAI
612-615	SAI
616-619	SAI
620-623	SAI
624-627	SAI
628-631	SAI
632-635	SAI
636-639	SAI
640-643	SAI
644-647	SAI
648-651	SAI
652-655	SAI
656-659	SAI
660-663	SAI
664-667	SAI
668-671	SAI
672-675	SAI
676-679	SAI
680-683	SAI
684-687	SAI
688-691	SAI
692-695	SAI
696-699	SAI
700-703	SAI
704-707	SAI
708-711	SAI
712-715	SAI
716-719	SAI
720-723	SAI
724-727	SAI
728-731	SAI
732-735	SAI
736-739	SAI
740-743	SAI
744-747	SAI
748-751	SAI
752-755	SAI
756-759	SAI
760-763	SAI
764-767	SAI
768-771	SAI
772-775	SAI
776-779	SAI
780-783	SAI
784-787	SAI
788-791	SAI
792-795	SAI
796-799	SAI
800-803	SAI
804-807	SAI
808-811	SAI
812-815	SAI
816-819	SAI
820-823	SAI
824-827	SAI
828-831	SAI
832-835	SAI
836-839	SAI
840-843	SAI
844-847	SAI
848-851	SAI
852-855	SAI
856-859	SAI
860-863	SAI
864-867	SAI
868-871	SAI
872-875	SAI
876-879	SAI
880-883	SAI
884-887	SAI
888-891	SAI
892-895	SAI
896-899	SAI
900-903	SAI
904-907	SAI
908-911	SAI
912-915	SAI
916-919	SAI
920-923	SAI
924-927	SAI
928-931	SAI
932-935	SAI
936-939	SAI
940-943	SAI
944-947	SAI
948-951	SAI
952-955	SAI
956-959	SAI
960-963	SAI
964-967	SAI
968-971	SAI
972-975	SAI
976-979	SAI
980-983	SAI
984-987	SAI
988-991	SAI
992-995	SAI
996-999	SAI



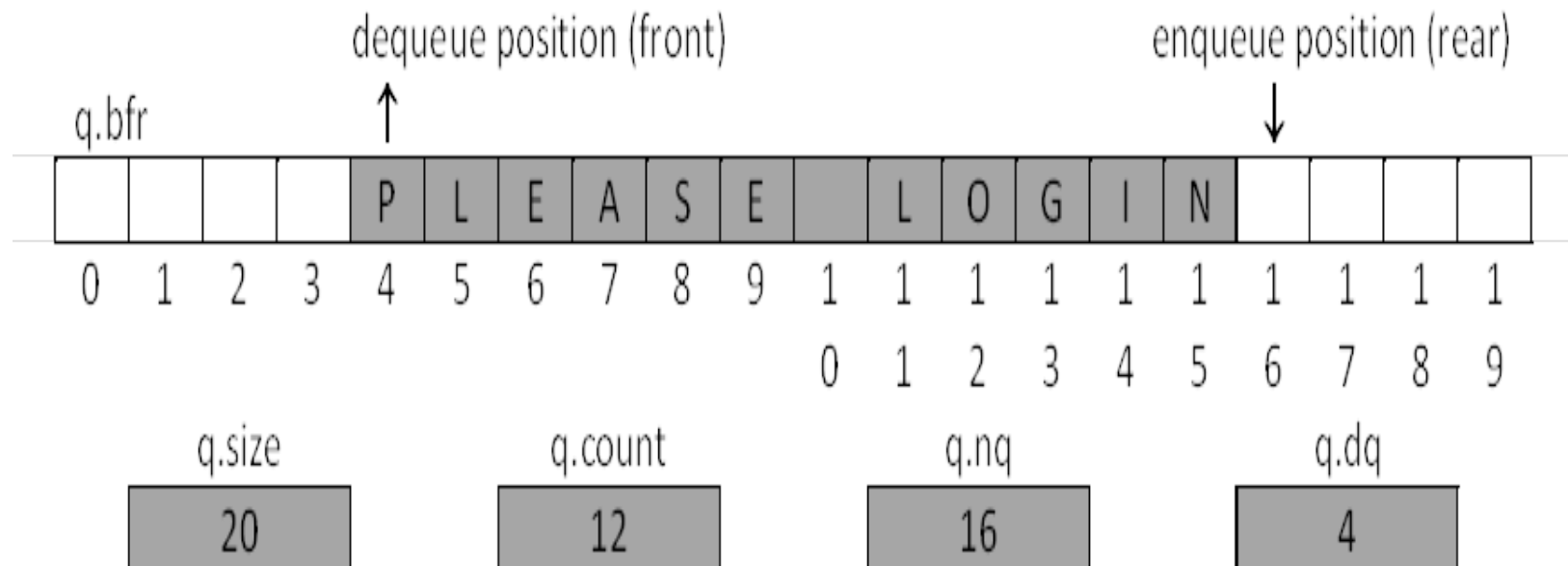
Bit in Interrupt
Registers

BUFFERING

Using a Queue to Decouple Data Arrival from Data Processing



Interrupt Buffering



Implementing a Queue in C

```
BOOL Enqueue(Queue *q, uint8_t data)
```

```
{
    BOOL full ;

    disable() ;
    full = q->count == q->size ;
    if (!full)
    {
        q->bfr[q->nq] = data ;
        if (++q->nq == q->size) q->nq = 0 ;
        q->count++ ;
    }
    enable() ;

    return !full ;
}
```

```
BOOL Dequeue(Queue *q, uint8_t *ptr2data)
```

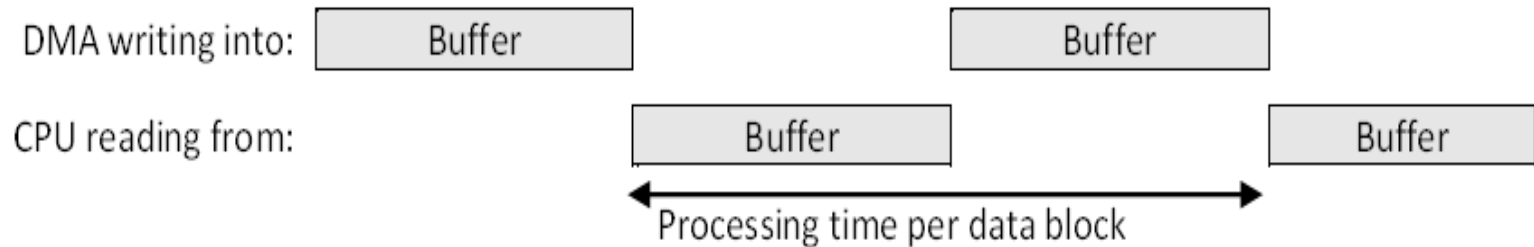
```
{
    BOOL empty ;

    disable() ;
    empty = q->count == 0 ;
    if (!empty)
    {
        *ptr2data = q->bfr[q->dq] ;
        if (++q->dq == q->size) q->dq
            = 0 ;

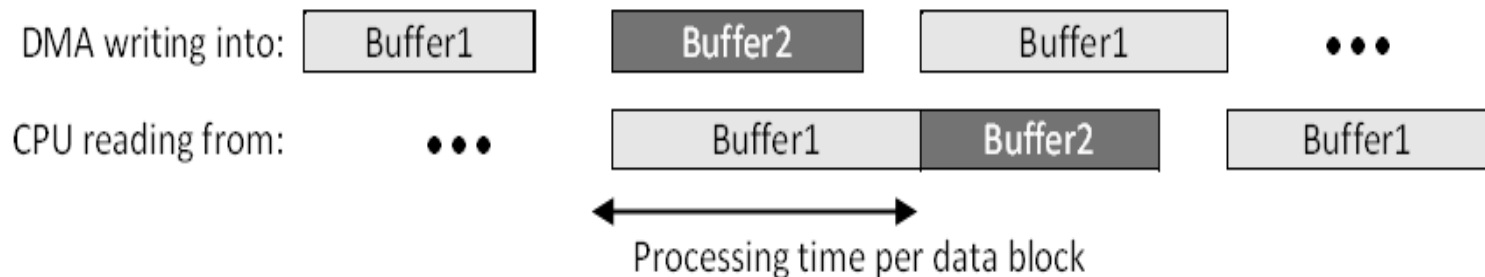
        q->count-- ;
    }
    enable() ;

    return !empty ;
}
```

Single Buffered DMA



Double Buffered DMA



SYNCHRONIZATION, TRANSFER RATE AND LATENCY

Polled Waiting Loops

- Test device status in a ***waiting loop*** before transferring each data byte.
- **Maximum data rate**: Determined by the time required to execute one iteration of the waiting loop plus the transfer.
- **Latency**: Unpredictable - no guarantee when the program will arrive at the waiting loop.

Synchronizing data transfers using polled waiting loops.

```
// TXFE (bit 7 of status port) =1 if transmitter holding register empty
```

```
#define TXFE      (1 << 7)
```

```
// RS232 Data Port = rs232 base address + 0
```

```
// RS232 Status Port      = rs232 base address + 24
```

```
void rs232Output(uint8_t *ptr2data, int bytes, volatile uint32_t *rs232base)
{
    for (int byte = 0; byte < bytes; byte++)
    {
        while ((*rs232base + 6) & TXFE) == 0)
        {
            // do nothing (wait for data to arrive)
        }
        *rs232base = (uint32_t) *ptr2data++;
    }
}
```

Polled waiting loop: Optimized in assembly.

```
EXPORT rs232Output  
rs232Output:
```

```
; R0 contains the parameter 'ptr2data'  
; R1 contains the parameter 'bytes'  
; R2 contains base address of RS232 device
```

```
Repeat:  CBZ      R1,Return    ; more data to send?  
OutWait: LDR      R3,[R2,#0x18] ; get status word  
         TST      R3,#0x80    ; TXFE = 1?  
         BEQ      OutWait     ; loop until ready  
         LDRB     R3,[R0],#1   ; get next byte & update pointer  
         STR      R3,[R2]     ; send data to device  
         SUB      R1,R1,#1    ; decrement byte count  
         B        Repeat      ; do it all again  
Return:  BX       LR          ; return
```

Cortex-M3 Instruction Clock Cycles¹

Instructions		Clock Cycles
PUSH, POP, LDM, STM		1 + #regs
SDIV, UDIV		2 - 12
SMLAL, UMLAL		4 - 7
SMULL, UMULL		3 - 5
Unconditional Branch (B, BL, BX)		2 - 4
Conditional Branch	Successful	2 - 4
	Failed	1
LDRD, STRD		3
ADR, MLA, MLS, & all LDR's and STR's		2
All other instructions		1

¹Assumes memory and processor run at the same speed.

Data Rate: Polled Waiting Loop

```
EXPORT rs232Output          ; Clock
rs232Output:                 ; Cycles

Repeat:  CBZ      R1,Return   ; 1 (Fail)
OutWait:  LDR      R3,[R2,#0x18] ; 2
          TST      R3,#0x80   ; 1
          BEQ      OutWait    ; 1 (Fail)
          LDRB     R3,[R0],#1  ; 2
          STR      R3,[R2]     ; 2
          SUB      R1,R1,#1    ; 1
          B        Repeat     ; 4
```

Clock cycles: 14

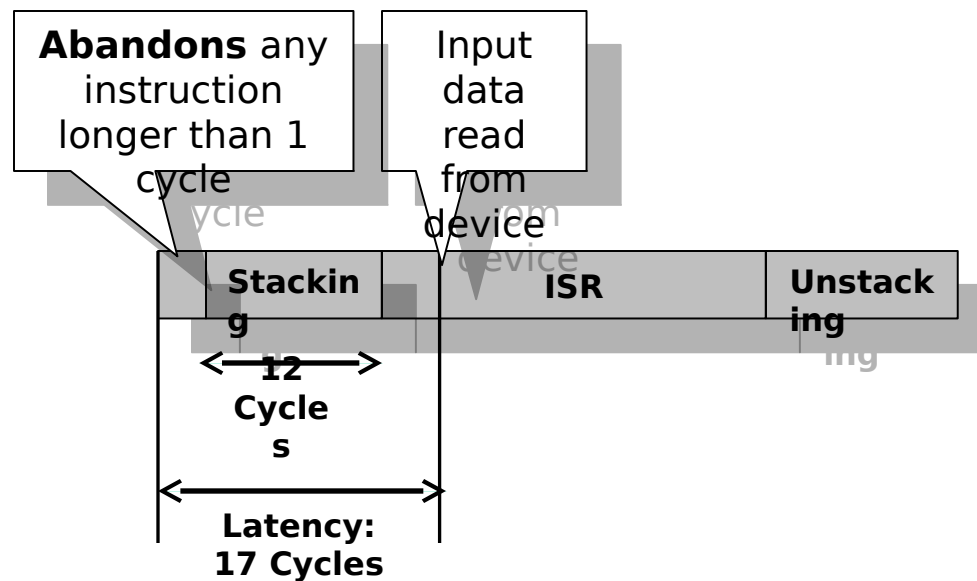
Execution time: $14 \times 20 \text{ nsec} = 280 \text{ nsec}$

Maximum Data Rate: $10^9 \div 280 = \mathbf{3.6 \text{ MB/sec}}$

Data Rate: Interrupt-Driven I/O

Rs232InputHandler:			Clock Cycles
ADR	R0,rs232InpDataPort ; R0 \equiv rs232 device address		2
LDR	R0,[R0] ; R0 \equiv rs232 input data byte		2
LDR	R2,rs232InputNQIndex ; update enqueue index		2
ADD	R2,R2,#1		1
AND	R2,R2,#0x3F ; increment & wrap (0 to 63)		1
STR	R2,rs232InputNQIndex		2
LDR	R3,rs232InputQueue ; R3 \equiv address of buffer		2
STRB	R0,[R3,R2] ; store data into buffer		2
LDR	R2,rs232InputCount ; update #items in queue		2
ADD	R2,R2,#1		1
STR	R2,rs232InputCount		2
BX	LR ; return		4
Tail-chaining time between successive ISRs:			6
Clock cycles:			29
Execution time:			$29 \times 20 \text{ nsec} = 580 \text{ nsec}$
Maximum Data Rate:			$10^9 \div 580 = \mathbf{1.7 \text{ MB/sec}}$

Interrupt Latency



Total latency: 17 Cycles × 20 nsec = **340 nsec**

Direct Memory Access

- Requires additional hardware to control data transfers independent of CPU.
- Competes with CPU for control of memory.
- Does not have to wait for current instruction to complete – only the current memory cycle.
 - Latency is thus 1 memory cycle.
 - If memory and I/O are on separate chips: 1 cycle = **60 nsec**
- Data Rate determined by memory speed.
 - 32-bit data bus: 4 bytes / (60×10^{-9} sec) = **66 MB/Sec**

Performance Estimates and Relative Cost

	Polled Waiting Loop	Interrupt- Driven I/O	Direct Memory Access
Maximum Transfer Rate	~3.6 MB/sec	~1.7 MB/sec	~66 MB/sec
Best-Case Latency	unpredictable	340 nsec	~60 nsec
Hardware Cost	Least	Low	Moderate
Software Complexity	Low	Moderate	Moderate