Interrupts (resumed)

Last time

- Exceptional control flow
  - (low-level mechanisms)

Today

- Using interrupts as client
  - Configure, enable, attach handler
- Coordination of activity
  - Exceptional and non-exceptional code, multiple handlers
  - Data sharing, writing code that can be safely interrupted

Looking ahead

Last regular week: Lab 7, Assign 7

Nab that complete system bonus!

Labs 8 & 9 will be devoted to group work on final project

Start brainstorming final project ideas now

Form teams (2 people is best)

Interrupted control flow

```
static int gcount;
```

```
void update_screen(void)
{
  console_clear();
  for (int i = 0; i < 10;
     console_printf("%d",
}
```

```
bool button_pressed(unsigned int pc)
{
  if (gpio_check_and_clear_event(BUTTON)) {
     gcount++;
     return true;
   }
   return false;
}
```

23 23 23 23 23
23 23 24 24 24

Questions on mechanics of suspend/transfer/resume?

Three layers to enable

1. Enable detection of specific event
   - For example, when we detect a falling clock edge on GPIO_PIN3 (PS/2 CLK)
2. Enable event as source of interrupts
   - E.g., GPIO interrupts
3. Enable global interrupts

Interrupt fires if and only all three are enabled

Forgetting to enable one is a common bug

Attach handler

Every interrupt calls same **`interrupt_vector`** function

- But we want different processing for mouse event versus key event versus timer event…

`interrupt_vector` designed as *dispatcher*:

- Client passes function pointer to "attach" as interrupt handler
- Client's function added to list of handlers
- On interrupt, Interrupt_vector calls each handler in list
- A hander responsible for determining if this is "its" interrupt
  - If so, process it, clear state, return true
  - Otherwise do nothing, return false
- Processing stops at first handler that reports interrupt has been handled
- Review our code in **`interrupts`** module

code/armtimer-blink

Detecting GPIO events

Register pin for event detection

- Many options: falling edge, rising edge, high level, etc.

Read/clear status in event detect register

- Bit is set when an event on the given pin occurs
- Must clear event bit or will re-trigger interrupt!

References

- P. 96-99 in BCM2835 ARM Peripherals doc
- Review our code in **`gpioextra`** module

A most frustrating page...

BROADCOM.

BCM2835 ARM Peripherals

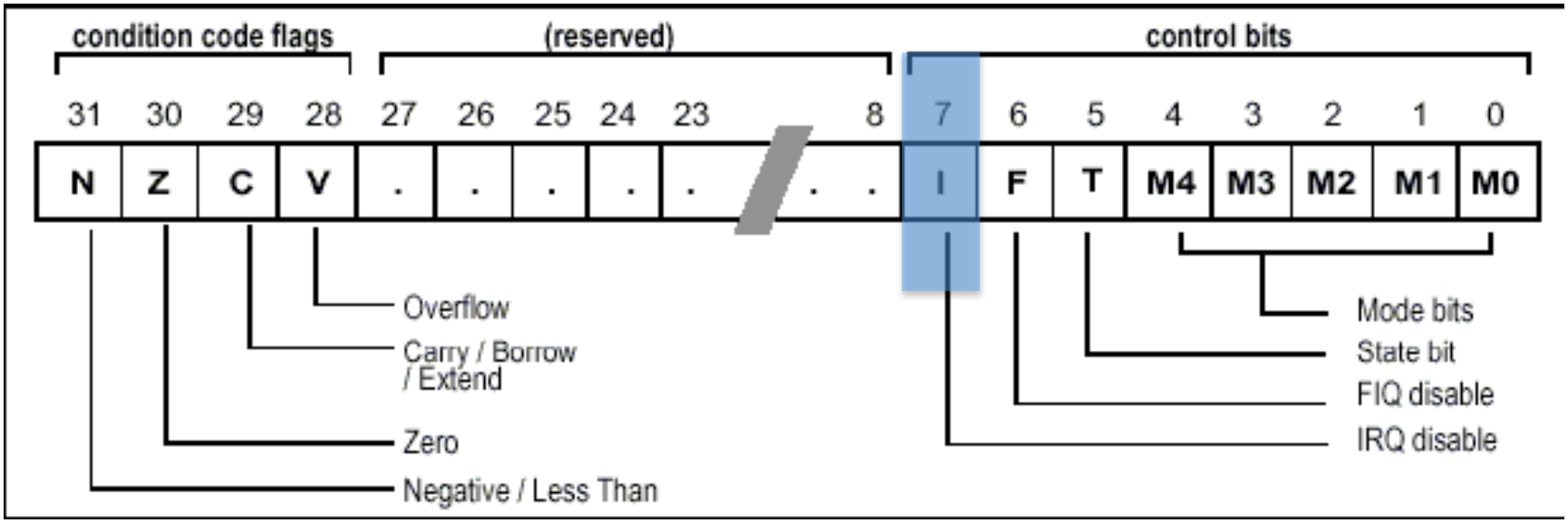
ARM peripherals interrupts table.

#	IRQ 0-15	#	IRQ 16-31	#	IRQ 32-47	#	IRQ 48-63
0		16		32		48	smi
1		17		33		49	gpio_int[0]
2		18		34		50	gpio_int[1]
3		19		35		51	gpio_int[2]
4		20		36		52	gpio_int[3]
5		21		37		53	i2c_int
6		22		38		54	spi_int
7		23		39		55	pcm_int
8		24		40		56	
9		25		41		57	uart_int
10		26		42		58	
11		27		43	i2c_spi_slv_int	59	
12		28		44		60	
13		29	Aux int	45	pwa0	61	
14		30		46	pwa1	62	
15		31		47		63	

Huh??

The table above has many empty entries. These should not be enabled as they will interfere with the GPU operation.

Enabling global interrupts




```
interrupts_global_enable:
    mrs r0, cpsr
    bic r0, r0, #0x80   @ clear I=0 enables IRQ interrupts
    msr cpsr_c, r0
    bx lr

interrupts_global_disable:
    mrs r0, cpsr
    orr r0, r0, #0x80   @ set I=1 disables IRQ interrupts
    msr cpsr_c, r0
    bx lr
```

code/button-interrupts

We're done!

Correct transfer of control to/from interrupt

- Assembly to save registers, state
- Call into C code
- Assembly to restore registers, resume

Interrupt vector installed in correct location

- Embed addresses with table so jumps are absolute
- Copy interrupt table to 0x0 in cstart

Enable and disable interrupts

- Specific interrupts, per-peripheral interrupts, global interrupts

Not quite

An interrupt can fire at any time

- Interrupt handler adds a PS/2 scan code to a queue
- Could do so right as main() is trying to pull a scan code out of the same queue
- Need to maintain integrity of shared queue

<u>Must write code so that it can be safely interrupted</u>

Atomicity

main code

```
static int nevents;

    nevents--;
```

interrupt handler

```
static int nevents;

    nevents++;
```

Q. What is the atomic (i.e., indivisible) unit of computation?

Q. Can an update to nevents be lost when switching between these two code paths?

A problem

main code

```
static int nevents;

    nevents--;
```

```
8074: ldr  r3, [pc, #12]
8078: ldr  r2, [r3]
807c: sub  r2, r2, #1
8080: str  r2, [r3]

8088: .word  0x0000a678
```

interrupt handler

```
static int nevents;

    nevents++;
```

```
808c: ldr  r3, [pc, #12]
8090: ldr  r2, [r3]
8094: add  r2, r2, #1
8098: str  r2, [r3]

80a0: .word  0x0000a678
```

How can an increment be lost if interrupt occurs here?

A problem

main code

```
static int nevents;

    nevents--;


8074: ldr  r3, [pc, #12]
8078: ldr  r2, [r3]
807c: sub  r2, r2, #1
8080: str  r2, [r3]

8088: .word  0x0000a678
```

interrupt handler

```
static int nevents;

    nevents++;


808c: ldr  r3, [pc, #12]
8090: ldr  r2, [r3]
8094: add  r2, r2, #1
8098: str  r2, [r3]

80a0: .word  0x0000a678
```

Instruction uses value copied into r2; increment of global by interrupt code is lost

Will volatile solve this?

Disabling interrupts

main code

```
interrupts_global_disable();
nevents--;
interrupts_global_enable();
```

interrupt handler

```
nevents++;
```

Q. Does increment need bracketing also?

Preemption and safety

Very hard, lots of bugs.

You'll learn more in CS110/CS140.

Two simple answers

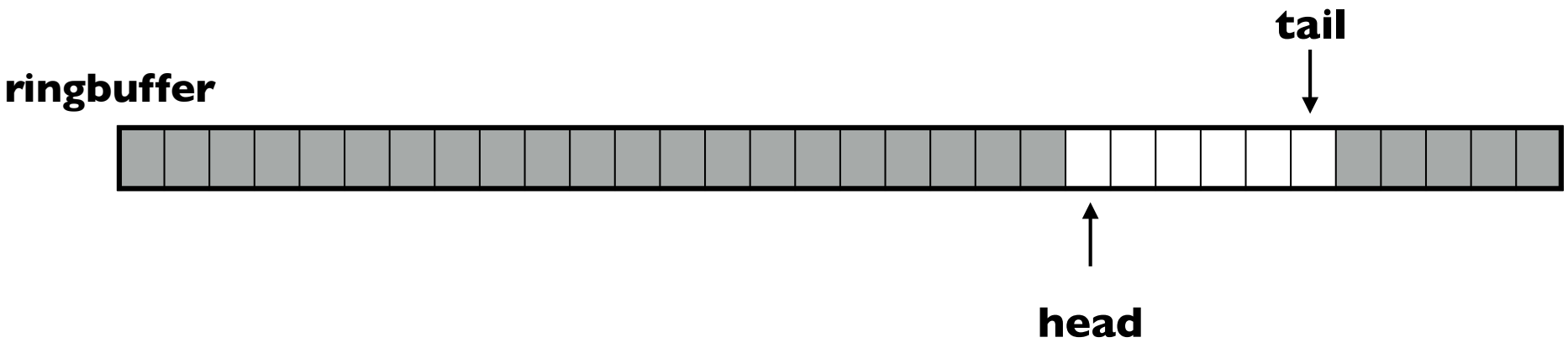
1. Use simple, safe data structures
   - write once, but not always possible
2. Otherwise, temporarily disable interrupts
   - always works, but easy to forget

Safe ringbuffer

A simple approach to avoid interference is for different code paths to not write to same variables

Queue implemented as ring buffer:

- Enqueue (interrupt) writes element to tail, advances tail
- Dequeue (main) reads element from head, advances head



Ringbuffer code

```
bool rb_enqueue(rb_t *rb, int elem)
{
     if (rb_full(rb)) return false;

     rb->entries[rb->tail] = elem;
     rb->tail = (rb->tail + 1) % LENGTH; // only writes tail
     return true;
}
```

```
bool rb_dequeue (rb_t *rb, int *elem)
{
     if (rb_empty(rb)) return false;

     *elem = rb->entries[rb->head];
     rb->head = (rb->head + 1) % LENGTH; // only writes head
     return true;
}
```

C-mastery carryover

Code style

Reading code to learn

Tuning your development process