ECE 271 – Microcomputer Architecture and Applications Lecture 15

Vince Weaver

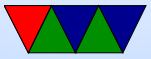
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

14 March 2019

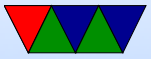
Announcements

- Read Chapter 11
- Midterm will be graded soon, not before break though
- Remember this was catch-up Lab week



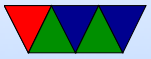
Lab #7 Preview

- Access the timer, in C



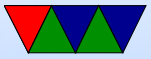
Timers

- Just a counter. Set up a clock input, then count how many times it happens.
- Can say, set up a 1MHz clock, then set it to count down from 1000, when it hits 0 it's been 1ms
- How do you tell when it hits 0?
- (Things get way more complicated, will cover after break)



Interrupts

- What do you do if you want to see if hardware is ready? Say the joystick pressed?
- Just have a loop that constantly checks? (polling)
- What's the downside of polling? (wastes resources, have to constantly be checking, what if you are busy and miss something, waste energy/power in tight loop)
- What is the alternative?



Interrupts

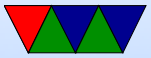
- When some sort of event happens, it "interrupts" what the CPU is doing
- Types of interrupts
  - Internal to CPU

    Timer

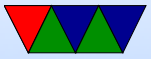
    Illegal instruction

    Illegal memory access
  - Something your code does

    System call

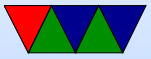


- Something external hardware does
  Keyboard press
  Network packet comes in
  Close lid of laptop
  Sound device wants more input



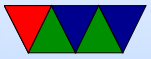
Interrupt Handling

- What happens when an interrupt comes in?
- Jumps to an "interrupt handler" that takes over
- Problem: how do we get back to our original code?
- Problem: can the handler use registers? Could it just save them on the stack? (does the stack pointer have to be valid or have room on it?)
- Cortex M does this differently from other ARM processors which always throws me because I am used to doing this on Raspberry Pis



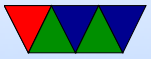
Interrupt Vectors

- Cortex-M has a set of vectors at bottom of memory (you can move it around with VTOR register)
- Address 0 is the value that gets put in the stack pointer at boot
- Address 4 through first 1k) are interrupt vectors
- A vector table is an array of addresses. Each interrupt source has a unique number (on Cortex M, for complex reasons, it is -15 to 240)
- Add 15 to source, then multiply by 4 (Remember, 32-



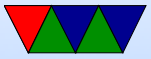
bits) then offset from 4 (to skip stack). Grab that value, and that's the address to jump to at an interrupt

- Look up the values
  - -15 to -1 are internal processor interrupts, 0 to 240 are external
  - -15 is Reset, (at address 4) which is called at boot or press black button
  - -1 is SystemTick (For lab)
  - Various hardware are the upper ones



Interrupt Stacking/Unstacking

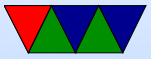
- Before jumping, it saves R0,R1,R2,R3 and R12,LR,PSR,PC to the stack.
- Which stack?
  There are actually *two* stack pointers, the MSP (main stack pointer) and PSP (process stack pointer)
  I think for this class we can assume it's always going to the MSP
- This could be worse, non cortex-M ARM processors have like 7 different modes each with it's own set of banked



registers

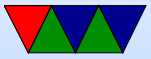
- At end, restores these all. Also clears the I flag
- Return by BX LR, the processor knows to do extra stuff if you are in interrupt handler mode

  Deep down it's actually putting a special magic value like 0xf00000XX in LR and returning to that, but that's not important for this class



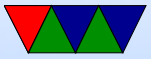
Interrupt Handler

- On Cortex-M can just be a plain C function (this isn't always true, other architectures require some assembly)



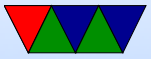
Nested Vectored Interrupt Controller (NVIC)

- 256 interrupts
- Each has 6 bits (spread across different regs)
  - Enable (ISER0..ISER7)
  - Disable
  - Pending
  - Un-pending
  - Active
  - SW trigger



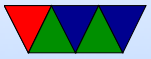
Interrupt Priority

- Can interrupt an interrupt. Why?
- Real time systems
- Non-maskable interrupt?



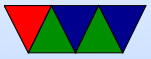
Enabling an Interrupt

- Setup handler
- Make sure vector points to it
- Enable the interrupt for the device you want in ISER
- Globally enable interrupts for the processor

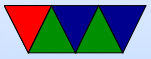


Handling an Interrupt

- Handler called.
- Save regs? Not on Cortex-M, does it for you
- Figure out what interrupt happened (Cortex-M a vector for each so not a problem?)
- "ACK" the interrupt, tell the hardware we are handling it
- Do whatever we want. Should this take a while?
- Exit interrupt and return to where we were. Cortex-M will clear interrupt flag and restore regs from special

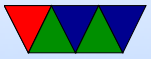


stack for is



Lab#7 Preview

- Setup a timer that operates at 1ms (1000Hz)
- Setup an interrupt handler that runs at 1000Hz, that decrements a value you set down to zero.
- Create a delay function that uses this timer to do "exact" timing.
  Set the value to 1000, then spin waiting for it to be zero.
- Use it to blink LED exactly
- Measure this with oscilloscope (know how to, taking 214?)



Apple II Timer/Interrupt Demo

Thwarted by a loose audio cable.