Welcome BackBack BackBack BackBack to CS429H!

Week 6

Ed meme recap:

don't write bugs.

anRtaHU6Ly9ka2F4ZS56Z3FnZXcuZW9mL3Z0aW93L3cvMS9maGRmZWtrLzFPX3pqN3lEdGNOdVhRNzU4cUwwb3N4YU92UnVhdlB3Xw==
#345

J Jocelyn Zhang STAFF
16 hours ago in Resources

ENDORSED UNPIN STAR WATCH 56 VIEWS

👀

9 Comment Edit Delete Unendorse ···

Sort by Newest

Questions on lecture content?
Or about cats?

Stress

- 429H is not an easy class
    - Lots of new materials
    - Unfamiliar programming environments
    - Fast, often relentless pace
- Struggling in this course is normal
    - There will be times you won't know the answer of the solution
    - This is expected—we want we everyone to succeed, but the only way we can help is if you ask for it
- If you find yourself overly overwhelmed or spending more time on this class than you think you should be, please reach out to Dr. Gheith or the TAs
    - We can help out as far as the class goes
    - We can provide other resources where we are not able to help

Mental health resource available at UT

P5

Poll

How's your status on P5?

A. What's P5?
B. I've heard of it
C. I've cloned the starter code and/or looked through it
D. I've started planning/writing code
E. I'm mostly done but might still have bugs
F. P5 any% speedrun

PSA, run your code with Gheith's GCC version

We will use this version with -O3 to grade.

The instructions:

https://edstem.org/us/courses/53774/discussion/4470800

P6

First, what's a subroutine?

- Is it the same thing as a function?
- Why is the name **sub**routine significant?
- How do we keep track of the execution state when going between different subroutines?

What’s a Coroutine? Something ducks walk on?

- Consider an ordinary function
  - May make calls to subroutines, but always executes sequentially
  - Can only run one of these at a time, given 1 CPU
- What if we want to run a lot of functions, interleaving their execution?
  - Need a notion of a “suspendable function”---a function that we can stop and resume
  - This way, we can make the PC jump between routines
  - **Concurrent** but not parallel
- How do we decide when to switch routines?
  - Preemptive - somehow force the CPU to switch tasks
  - **Cooperative** - define specific “yield points” at which the CPU switches tasks

What's a Coroutine? Something ducks walk on?

- Consider an ordinary function
  - Uses function parameters and return values to exchange information
- What if we want separate coroutines to communicate?
  - Need an explicit mechanism ~~(one that says bad words a lot)~~
  - Useful abstraction: **channels**
- Channels
  - Object passed to one or more routines
  - Interface consists of **send** and **receive** functions
    - What should these do?
    - What data structure does this sound like?

Fun Activity Time!

- Groups of 2 will pretend to be coroutines.
- One coroutine gets to execute at a time.
- Coroutines will send each other messages. A send cannot finish unless the receiver is also trying to receive at the same time. In this case, the send will **block** the sender from executing. Ditto for receives.
- When a coroutine **yields**, any unblocked coroutine gets the chance to run. The TAs will use a tiebreaker to see who gets to run, and the rest of the class also helps in this tiebreaker.
- For the purposes of this activity, sends and receives will always yield.
- The coroutines only get to look at their own instructions, everyone else can look at the entire program. It's posted on Ed now!

Coroutines in Practice - Saving Progress

- Consider an ordinary function
  - All necessary information for execution and returning are saved on the stack and registers.
- With more than one routine, we must save this information
- How do we keep track of every routine’s state?

What is a stack?

What is a stack in computers?

What are stacks good for?

- Storing temporary/local variables
- Storing return addresses for function calls
- Storing function arguments

More on stacks

- Where is the stack usually?
- How do you find it?
- Does it have to be this way?

P6 - Some Assembly Required

- Context switching via magic.S
  - takes the active (running) coroutine and swaps it for a different coroutine, saving the first one's state after deactivating it
  - general flow of a context-switching function:
    - Record the complete state of the currently running coroutine and write it (implicitly or explicitly) into a **Routine (CCB)** struct
    - Read the state of the next coroutine from its **Routine** struct, and restore it
    - return (but to where????)
  - state in the **Routine** struct can be saved explicitly (through struct fields) or implicitly (by pushing to the stack and saving only the stack pointer in the Routine struct

Poll

Which ISA will P6 use?

1. ARM
2. x86
3. RISC-V
4. PowerPC
5. MIPS

P6 - Some Assembly Required

- Calling Convention:
  - X86:
    - Args go in %rdi, %rsi, %rdx, %rcx for integral types (in that order).
      - You will not need more than that many arguments
    - Return value goes in %rax
    - Callee saved registers: %rsp, %rbp, %rbx, %r12, %r13, %r14, %r15
  - ARM:
    - Argos go in x0, x1, x2, x3 for integral types (in that order).
      - You will not need more than that many arguments
    - Return value goes in x0
    - Callee saved registers: x19-x28

P6 - Some Assembly Required

- Where might you need to return a value from assembly?
- Which registers should you save? Caller/callee/both?

P6 - Some Assembly Required

- how can we tell the compiler that there is an asm function called magic?
  - `extern <return_type> magic(<args>...);`
- what do we do in assembly to create the function definition?
  - `.global magic`
  - `magic:`
  - `    ret`
- similar method to achieve the opposite effect
  - `.extern c_function`
  - `call c_function / bl c_function`

Coroutines in Practice - The Real World

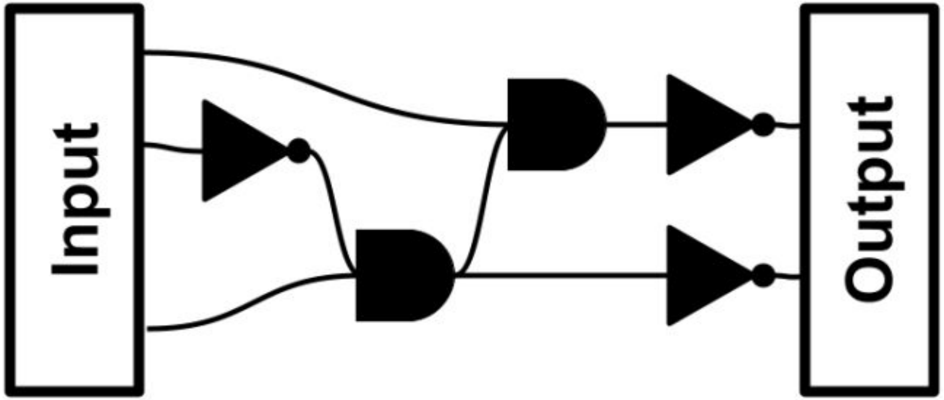
- "lazily evaluated" computations
  - Generators in Python are an example of this
- simpler iterator implementations
  - Remember writing an iterator for Boggle in 314H? How could it be easier with coroutines?
- asynchronous programming
  - When you have to wait for the network, disk, etc., but want to keep doing things in the meantime, coroutines can make that really nice and simple

Quiz everyone say YIPPEE!

REVIEW

Question 1

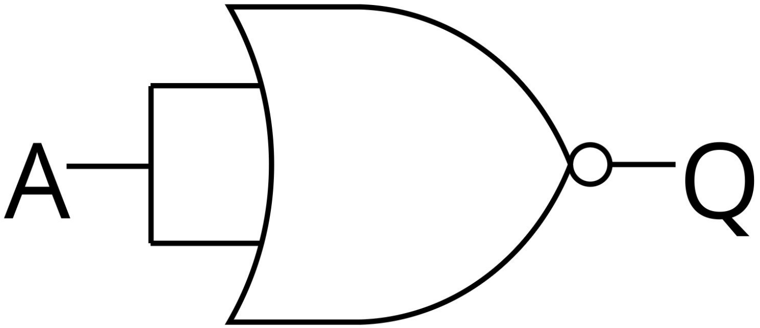
[1 point] The concept of settling time is how long a logical gate takes to converge to an output after receiving an input. After a gate "settles", we can reliably read the output of the gate. If each NOT gate takes 5 ns to settle and each AND takes 7 ns to settle, evaluate the time it takes for the circuit below to settle on its output values.



Question 2a

creating gates with nor

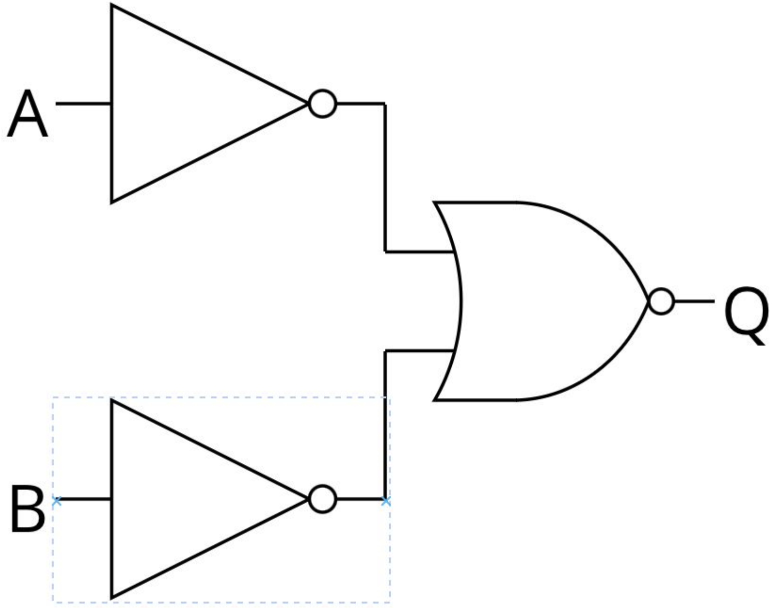
not:



Question 2b

creating gates with nor

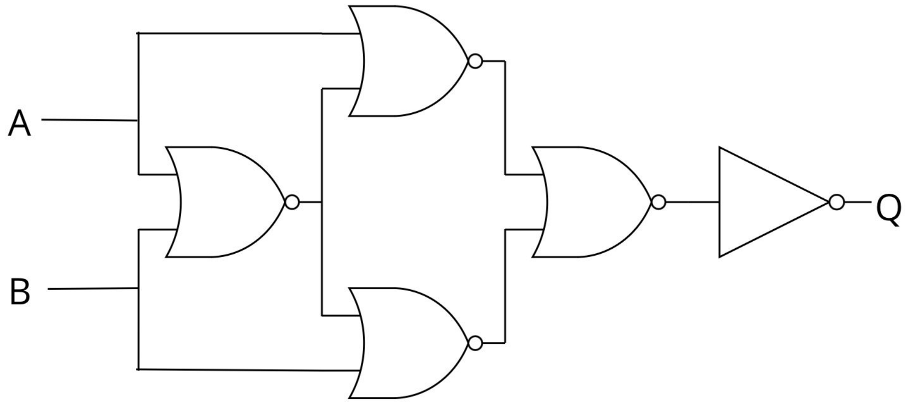
and:

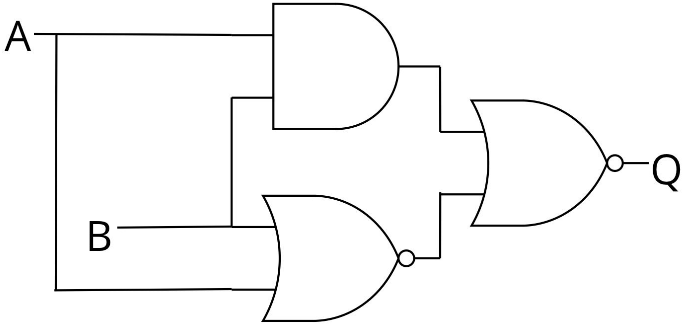


Question 2c

creating gates with nor

xor:





Question 3

[0.5 points] What is your favorite instruction in ARM? Briefly explain.

- If you were creating a rubric for this question, what would you do?
- What would be required for full credit?

Question 3

[0.5 points] What is your favorite instruction in ARM? Briefly explain.

- If you were creating a rubric for this question, what would you do?
- What would be required for full credit?

“B is my favorite instruction because it is fast to type with just one character”

- Y’all did well on this

Question 4

[1 point] The `CSETEQ` instruction on ARM will set a register to 1 if the zero flag is set (the previous comparison resulted in equality), or 0 otherwise. Implement the functionality of this instruction using only (conditional or unconditional) branches and move statements: fill in the following code and produce the output in the register `x7`.

Example of CSETEQ:

```
// Assume x0 = 3, x1 = 4, x2 = 4
cmp x1, x0
cseteq x3  // x3 is now 0
cmp x1, x2
cseteq x4 // x4 is now 1
```

Answer space:

```
cmp x5, x6
// implement cseteq x7 here
```

Question 4

[1 point] The `CSETEQ` instruction on ARM will set a register to 1 if the zero flag is set (the previous comparison resulted in equality), or 0 otherwise. Implement the functionality of this instruction using only (conditional or unconditional) branches and move statements: fill in the following code and produce the output in the register `x7`.

```
cmp x5, x6
b.eq equals
mov x7, #0
b end
equals:
mov x1, #1
end:
```

```
if (x5 != x6) {
    x7 = 0;
} else {
    x7 = 1;
}
```

Question 4

[1 point] The `CSETEQ` instruction on ARM will set a register to 1 if the zero flag is set (the previous comparison resulted in equality), or 0 otherwise. Implement the functionality of this instruction using only (conditional or unconditional) branches and move statements: fill in the following code and produce the output in the register `x7`.

```
cmp x5, x6
mov x7, #0
b.ne end
mov x1, #1
end:
```

```
x7 = 0;
if (x5 == x6) {
    x7 = 1;
}
```

Question 5

bitstring $b_{-1}b_{-2}...b_{-(N-1)}b_{-N}$

value $b_{-1}*2^{-1} + b_{-2}*2^{-2} + ... + b_{-(N-1)}*2^{-(N-1)} + b_{-N}*2^{-N}$.

- Basic addition works the same as with regular binary numbers, adding bits from right to left and propagating carry values
- If you have N bits, numbers that are not of the form k/2^N where k is some integer cannot be represented by our system

Question 6

Two's Complement

- Arithmetic is simple
- Adding and subtracting are the same
- 2^N distinct values representable
- Negation is hard
- Sign extension is hard

Linner Linner Chicken Dinner

- Arithmetic is hard
- Adding and subtracting are not the same
- 2^N - 1 distinct values representable
- Negation is easy
- Sign extension is trivial

Question 6

Two's Complement

- Arithmetic is simple
- Adding and subtracting are the same
- 2^N distinct values representable
- Negation is hard
- Sign extension is hard

One's Complement

- Arithmetic is slightly harder
- Adding and subtracting are nearly the same
- $2^N - 1$ distinct values representable
- Negation is slightly less hard
- Sign extension slightly less hard

Question 6 (Bonus)

- a + b + carry bit
- Really, it’s as shrimple as that

Questions?

```
                          oooo$$$$$$$$$$$$oooo
                      oo$$$$$$$$$$$$$$$$$$$$$$$$o
                   oo$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$o         o$   $$ o$
   o $ oo        o$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$o       $$ $$ $$o$
oo $ $ "$      o$$$$$$$$$    $$$$$$$$$$$$$    $$$$$$$$$o       $$$o$$o$
"$$$$$$o$     o$$$$$$$$$      $$$$$$$$$$$      $$$$$$$$$$o    $$$$$$$$
  $$$$$$$    $$$$$$$$$$$      $$$$$$$$$$$      $$$$$$$$$$$$$$$$$$$$$$$
  $$$$$$$$$$$$$$$$$$$$$$$    $$$$$$$$$$$$$    $$$$$$$$$$$$$$  """$$$
   "$$$""""$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$     "$$$
    $$$   o$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$     "$$$o
   o$$"   $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$       $$$o
   $$$    $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$" "$$$$$$ooooo$$$$o
  o$$$oooo$$$$$  $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$   o$$$$$$$$$$$$$$$$$
  $$$$$$$$"$$$$   $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$     $$$$""""""""
 """"       $$$$    "$$$$$$$$$$$$$$$$$$$$$$$$$$$$"      o$$$
            "$$$o     """$$$$$$$$$$$$$$$$$$"$$"         $$$
              $$$o          "$$""$$$$$$""""            o$$$
               $$$$o                                o$$$"
                "$$$$o      o$$$$$$o"$$$$o        o$$$$
                  "$$$$$oo     ""$$$$o$$$$$o   o$$$$""
                     ""$$$$$oooo  "$$$o$$$$$$$$$"""
                        ""$$$$$$$oo $$$$$$$$$$
                                """"$$$$$$$$$$$
                                    $$$$$$$$$$$$
                                     $$$$$$$$$$"
                                      "$$$""""
```