CSO-Recitation 12

CSCI-UA 0201-007

R12: Assessment 10 & Combinational logic

Today’s Topics

- Assessment 10
- Lab-4
- Combinational logic
  - How to build a combinatorial logic circuit
  - MUX

Lab-4

Lab-4

- If you don’t do bonus
  - only need to write code in file `mm-implicit.c`
  - a very simple design -> malloc using implicit list without footer
    - Complete helper function next_chunk and use it to complete the heap checker function mm_heapcheck
    - Complete helper functions ask_os_for_chunk, split, first_fit and use them to implement mm_malloc
    - Complete helper functions payload2header, coalesce and use them to implement mm_free
    - Complete mm_realloc
- Tips
  - Please follow the instruction & comment code! – the lab shouldn’t be hard
  - Review helper functions, pseudocode, steps summary
- Due after Thanksgiving, encourage to do bonus part

realloc

- In the C, the **realloc** function is used to resize a block of memory that was previously allocated
- The realloc function allocates a block of memory (which be can make it larger or smaller in size than the original) and copies the contents of the old block to the new block of memory, if necessary
- void *realloc(void *ptr, size_t size);
  - if ptr is NULL, the call is equivalent to malloc(size)
  - if size is equal to zero, the call is equivalent to free(ptr)
  - if ptr is not NULL, it must have been returned by an earlier call to malloc or realloc

realloc

- Copies the contents of the old block to the new block of memory, if necessary
  - If the new block is larger, everything else is uninitialized
  - How to copy data?
    - See a C library function, memcpy()
- void *memcpy(void *dest, void * src, size_t n)
  - copies **n** characters from memory area **src** to memory area **dest**.

Combinational logic

Building Blocks

How we get there..

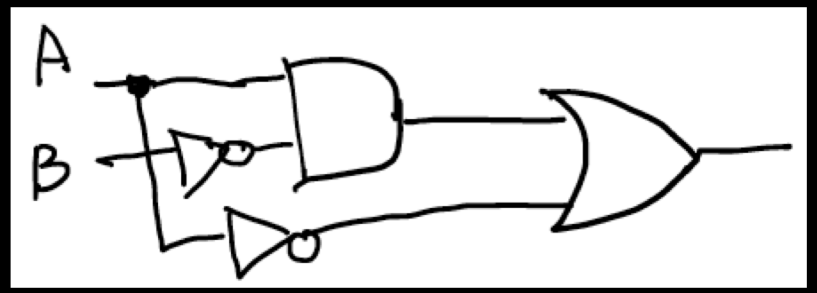
- Instruction set (e.g. X86) is the interface between software and hardware
- Logic design
  - assumes the existence of a collection of standard logic components
    - made from logic gates -> made from small groups of transistors
  - studies the organization of primitive logic components into subsystems that perform calculation and data transfer
    - e.g. An arithmetic-logic unit (**ALU**) is the part of a CPU that carries out arithmetic and logic operations on the operands in computer instruction words

Combinational Logic

- There is no memory
  - That is, the outputs are a function ONLY of the current inputs, not of anything in the past
- Values are either true or false (but not both, or anything else)
  - We commonly represent true with 1 and false with 0
- There is at least one input and at least one output
  - But there can be more
  - Each input is either true or false
  - Each output should be defined for all possible values for the inputs!

Combinational Logic

- There are three main ways to represent combinational circuits
  1. As a circuit diagram
  2. As a set of equations/expressions
  3. As a truth table



$$A\bar{B} + \bar{A}$$

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

Combinational Logic

- Logical expressions are build from a number of “gates”
- You are already familiar with the most important ones!
  - AND, OR, NOT
    - All boolean functions can be written with these three building blocks!
  - There are others, like XOR and NAND
    - All boolean functions can be written with just NAND!!!
      - e.g. NOT(A) == NAND(A,A); AND(A,B) == (A NAND B) NAND (A NAND B)
- Letters/words are used to represent the inputs

Combinational Logic

- Basic logic design
  - Logic circuits == Boolean expressions
- How to build a combinational logic circuit
  - Step1: Specify the truth table
  - Step2: Output is the sum of products

Filling in a truth table

- First, list out all of the possible inputs
  - For N inputs, there are 2^N possibilities
  - It helps to split things in two, that is, have space for the 2^N possibilities, and for the first variable put 0 for the first half and 1 for the second half
    - For the next variable, put 0 for the first half of the previous variable's 0 group and 1 for the second half, then do the same for its 1 group
- Then, evaluate the expression for each possible inputs
  - This is tedious, so there are some shortcuts, especially if the expression is written nicely

Filling in a truth table

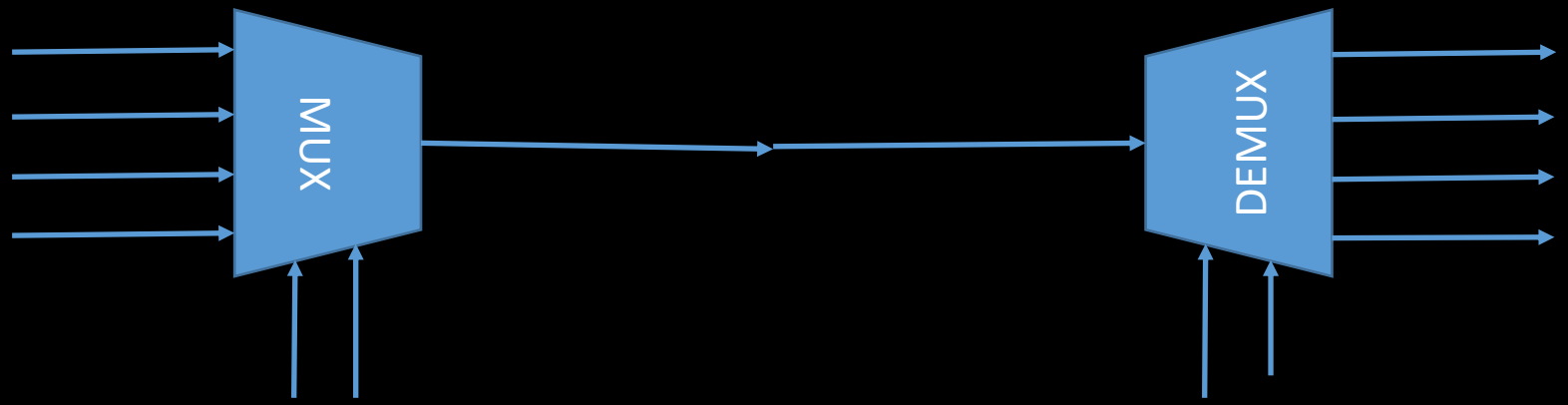
- Groups of things ANDed together are typically called a clause
  - That is, clauses are things separated by ORs
- For each clause, see what the variables present are
  - For any variable that is negated, keep in mind that that variable is 0
  - For any other variable in the clause, keep in mind it is 1
  - Then, look through the truth table and wherever you see a row that has all of the variables in the clause with the right value, put a 1 for the output
- Then when you are out of clauses, fill in 0 for any output left

Getting a circuit from a truth table

- Find the sum of product
- For each output, look for where that output is 1 in the truth table
  - Look at the list of inputs
    - anywhere an input a is 1, write a in the clause
    - Anywhere an input a is 0, write ~a in the clause
  - Say the output equals all of the clauses ORed together
  - Typically you can simplify, but that's a topic for a different day

Multiplexor (MUX)

- A multiplexor is a device which takes in multiple signals and outputs a single signal
- The purpose of using a multiplexer is to make full use of the capacity of the communication channel and greatly reduce the cost of the system
  - On the receiving side, a demultiplexer splits the single data stream into the original multiple signals



Multiplexor (MUX)

- In digital circuit design, the selector wires are of digital value
- 4-to-1 Multiplexor
- It can be noted that 2^N non-select signals require N select signals
  - 8-to-1 MUX, 16-to-1 MUX…

2^N input signals