CSO-Recitation 11

CSCI-UA 0201-007

R11: Assessment 10 & Combinational logic

Today’s Topics

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

Assessment 10

Q1 Implicit list

Suppose your implicit list design uses both header and footer. Both have the following type (Lecture slides 30):

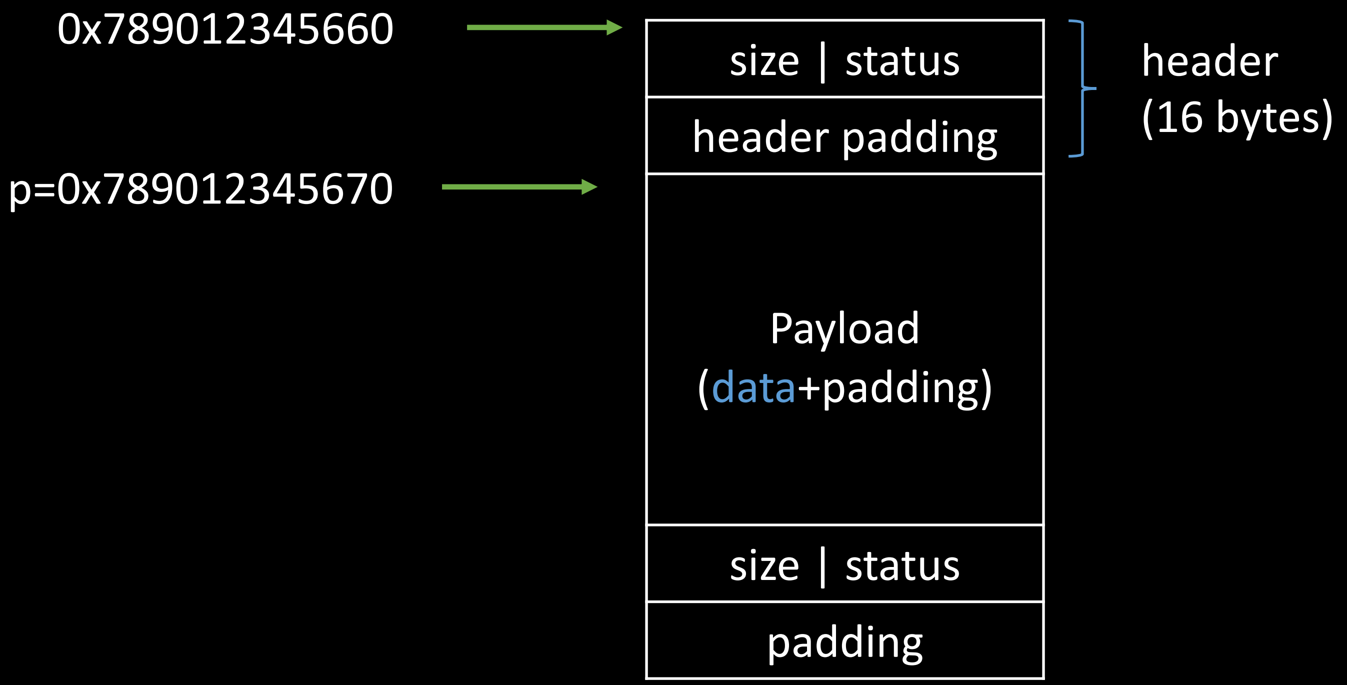
- get_status()
- get_size()
- set_size_status()
- set_status()
- set_size()
- payload2header()
- payload2footer()
- footer2header()
- curr2prev()
- …

Basic helper
Find in lecture slides

```
typedef struct {
  unsigned long size_and_status;
  unsigned long padding;
} header;
```

Q1.1 payload2header example

- Suppose a user invokes free(p) using pointer p whose value is 0x789012345670. What is the memory address for the start of the chunk that contains the allocated space (payload) that should be freed? (To faciliate autograding, please write your answer in hex with prefix 0x, ignoring leading zeros and using lowercase letters)
- 0x789012345660



Q1.2 payload2header

```
header* payload2header(void *p)
{
    header *h;
    ______???__________;
    return h;
}
```

`payload2header` takes as argument a pointer to the start of the payload in the chunk, and returns a pointer to the chunk's header.

Which of the following C statement to use for the missing line?

A. h = (header *)p - sizeof(header);

B. h = $\boxed{\text{(header *)}}$p - 1;

pointer arithmetic:
-1 ⇔ -sizeof(header) bytes

logic: h=p-(sizeof(header) *bytes*)

C. h = (header *)$\boxed{\text{((char *)}}$p - sizeof(header));

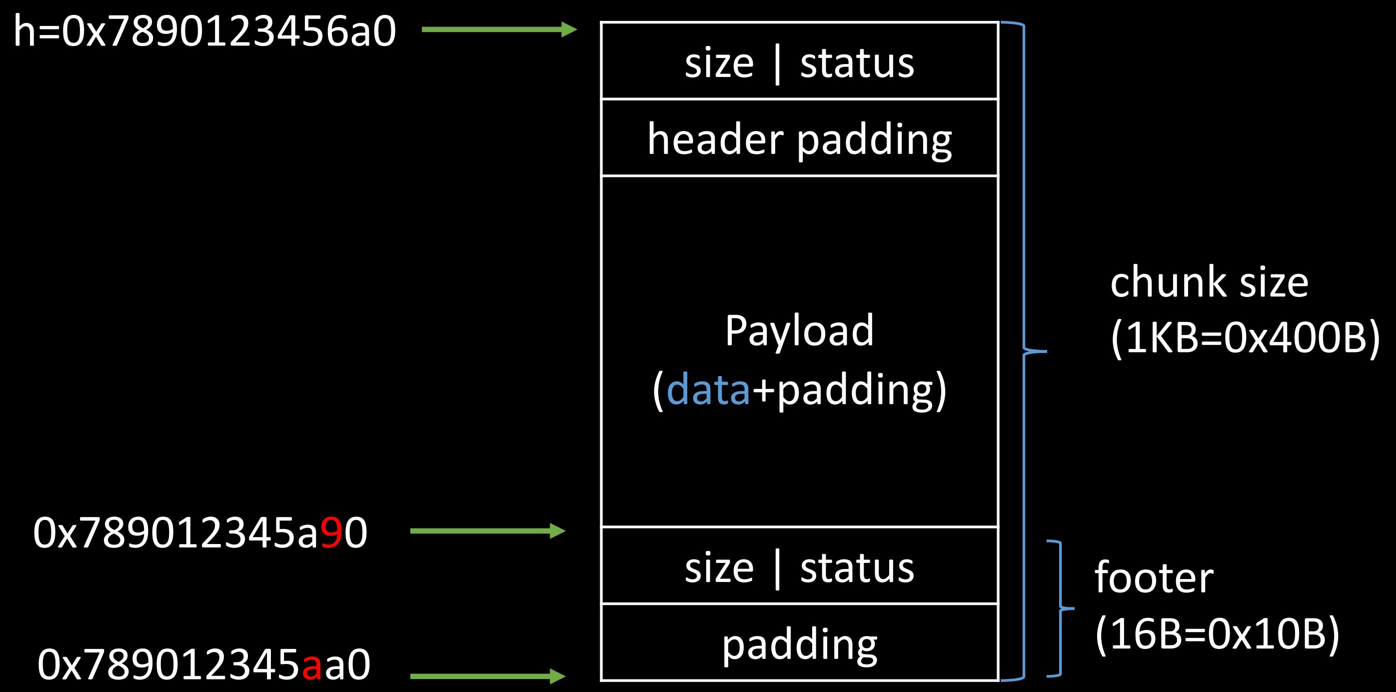
D. h = (char *)p - 1;

E. None of the above.

pointer arithmetic:
-1 ⇔ -sizeof(char)=1 bytes
-sizeof(header) ⇔ -sizeof(header) bytes

Q1.3 header2footer example

- Suppose pointer variable h points to the beginning of a chunk and has value 0x7890123456a0. If the total size of the chunk is 1KB (including header and footer fields), then what is the memory address for the footer of this chunk?
- 0x789012345a90



Q1.4 header2footer

```
header* header2footer(header *h)
{
    header *f;
    ______???___________;
    return f;
}
```

`header2footer` takes as argument a pointer to the start of the chunk, and returns a pointer to the same chunk's footer.

Which of the following C statement to use for the missing line? Note that `get_size` is a helper function that returns the chunk size encoded in the header/footer field size_n_status.

A. f = h + 1;

Q: how many bytes does it step forward?

B. f = h - 1;

C. f = h + get_size(h);

D. f = (header *)((char *)h + get_size(h));

E. f = h - get_size(h);

F. f = (header *)((char *)h - get_size(h));

cast to char: +1 <=> +1 byte

G. f = (header *)((char *)h + get_size(h) - sizeof(header));

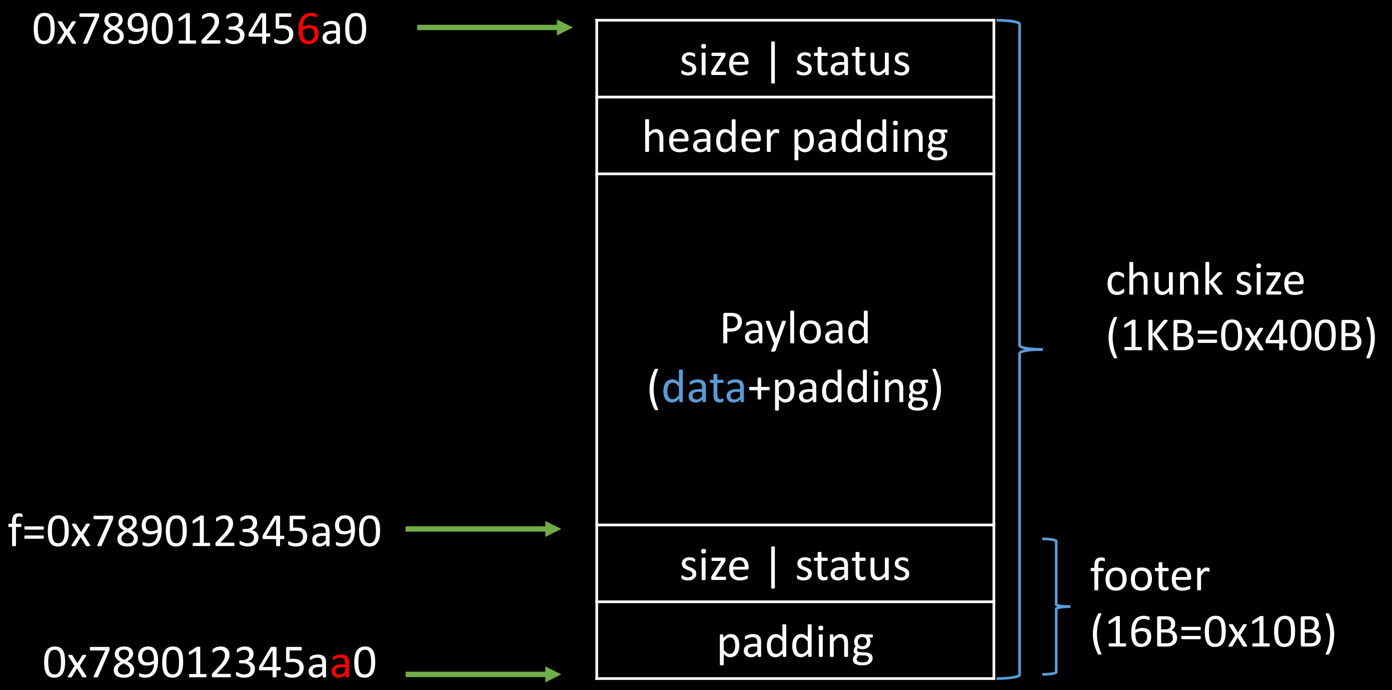
H. f = (header *)((char *)h - get_size(h) + sizeof(header));

I. None of the above.

logic: f=h
+(chunk_size bytes)
-(footer_size bytes)

Q1.5 footer2header example

- Suppose pointer variable f points to the beginning of a chunk's footer and has value 0x789012345a90. If the total size of the chunk is 1KB (including header and footer fields), then what is the memory address for the header of this chunk?
- 0x7890123456a0



Exactly the same as Q1.3

Q1.6 footer2header

```
header* footer2header(header *f)
{
    header *h;
    _____???__________;
    return h;
}
```

`footer2header` takes as argument a pointer to the footer of the chunk, and returns a pointer to the same chunk's header.

Which of the following C statement to use for the missing line? Note that `get_size` is a helper function that returns the chunk size encoded in the header/footer field size_n_status.

A. h = f + 1;

B. h = f - 1;

C. h = f + get_size(f);

D. h = (header *)((char *)f + get_size(f));

E. h = f - get_size(f);

F. h = (header *)((char *)f - get_size(f));

G. h = (header *)((char *)f + sizeof(header) - get_size(f));

H. h = (header *)((char *)f - sizeof(header) + get_size(f));

I. None of the above.

logic: h=f
+(footer_size bytes)
-(chunk_size bytes)

Q1.7 curr2prev example

- Suppose pointer variable h points to the beginning of some chunk and has value 0x789012345aa0. Suppose this chunk has size 4KB and its previous chunk has size 1KB. What is the memory address for the beginning of its previous chunk?
- 0x7890123456a0

- logic: h - size of the previous chunk
- 0x789012345aa0 – 1KB
- 0x789012345aa0 – 0x400 = 0x7890123456a0

Q1.8 curr2prev example

```
header* curr2prev(header *curr)
{
    header *prev_footer;
    _____???__________;
    return footer2header(prev_footer);
}
```

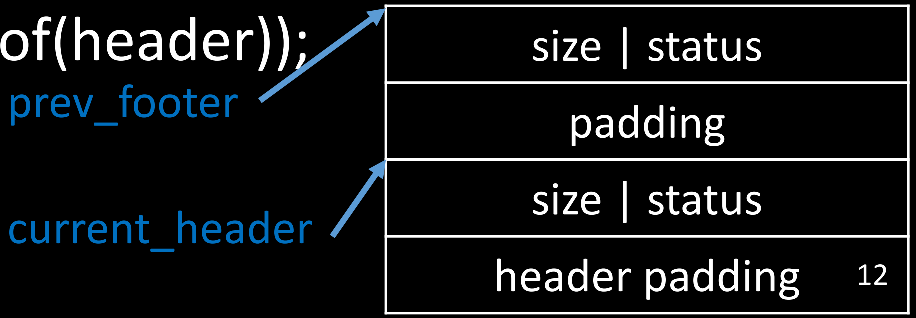
`curr2prev` takes as argument a pointer to the current chunk's header, and returns a pointer to the previous chunk's header.

Which of the following C statement to use for the missing line? Note that footer2header is the helper function that returns a pointer to the chunk's header given a pointer to the same chunk's footer.

A. prev_footer = curr -1 ;
B. prev_footer = curr - sizeof(header);
C. prev_footer = (header *)((char *)curr - sizeof(header));
D. prev_footer = curr - 2;
E. prev_footer = curr - 2*sizeof(header);
F. prev_footer = (header *)((char *)curr - 2*sizeof(header));
G. None of the above.

curr2prev
- curr_header2prev_footer
- prev_footer2prev_header -> footer2header

curr_header2prev_footer
- curr - (sizeof(header) bytes)
- ⇔ curr - 1



Q2 Explicit list

Which of the following statements are true about explicit list?

A. The explicit list design explicitly chains together all chunks of the heap into a linked list.

B. The explicit list design explicitly only chains together all free chunks of the heap into a linked list.

C. The explicit list design incurs more memory overhead than the implicit list design because it uses extra space in the header to store the next/prev fields.

D. malloc(...) in the explicit list design is faster than that of implicit list because it does not need to scan over allocated chunks.

E. free(...) in the explicit list design is faster than that of implicit list because it does not need to scan over allocated chunks.

Q2 Explicit list, Choice C

Explicit List
Allocated

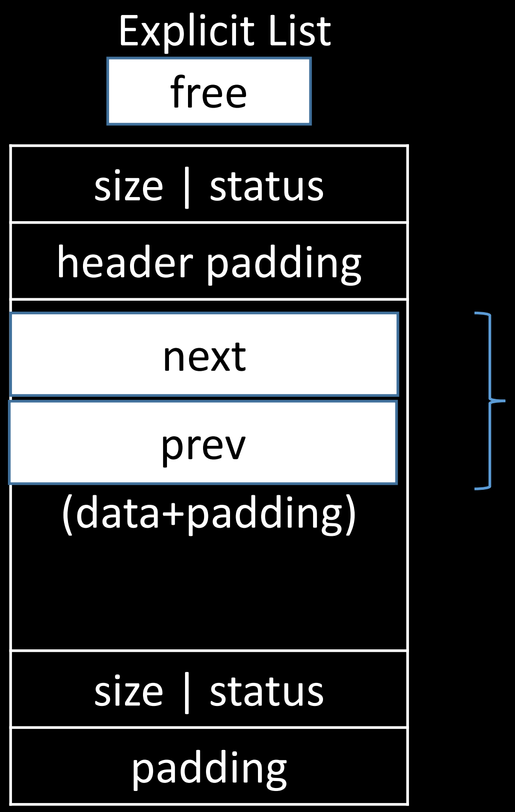
size | status
header padding
Payload (data+padding)
size | status
padding

No difference

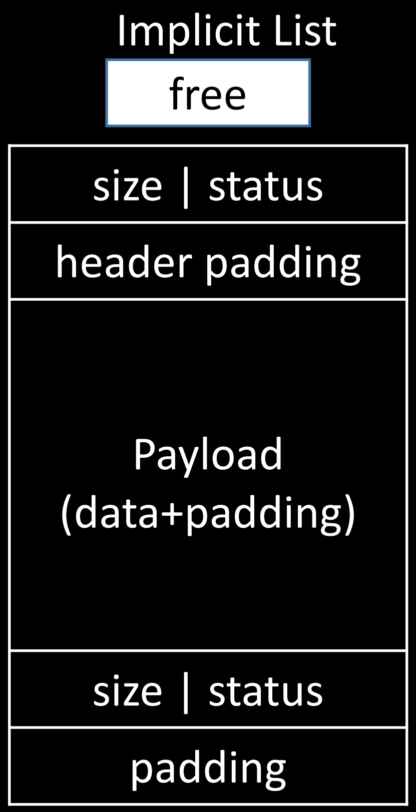
Implicit List
Allocated

size | status
header padding
Payload (data+padding)
size | status
padding

Q2 Explicit list, Choice C



- Explicit List reuses the payload space to store next/prev
- No extra space needed!



Q3 Buddy system

A special case of segregated list:
- each freelist has identically-sized blocks
- block sizes are power of 2

allocate:
- Recursive split in half

free:
- Recursively merge

Which of the following statements are true about the buddy system?

A. All chunks have sizes that are powers-of-2.

B. During free(...), coalescing only happens once by merging the freed chunk with its buddy of the same size if the buddy is free.

C. During free(...), coalescing is done recursively by repeatedly merging the freed chunk with its buddy of the same size and repeating the merge process for the resulting larger free chunk until its buddy is no longer free.

D. The design maintains multiple free lists each of which contains free chunks of the same (powers-of-2) size.

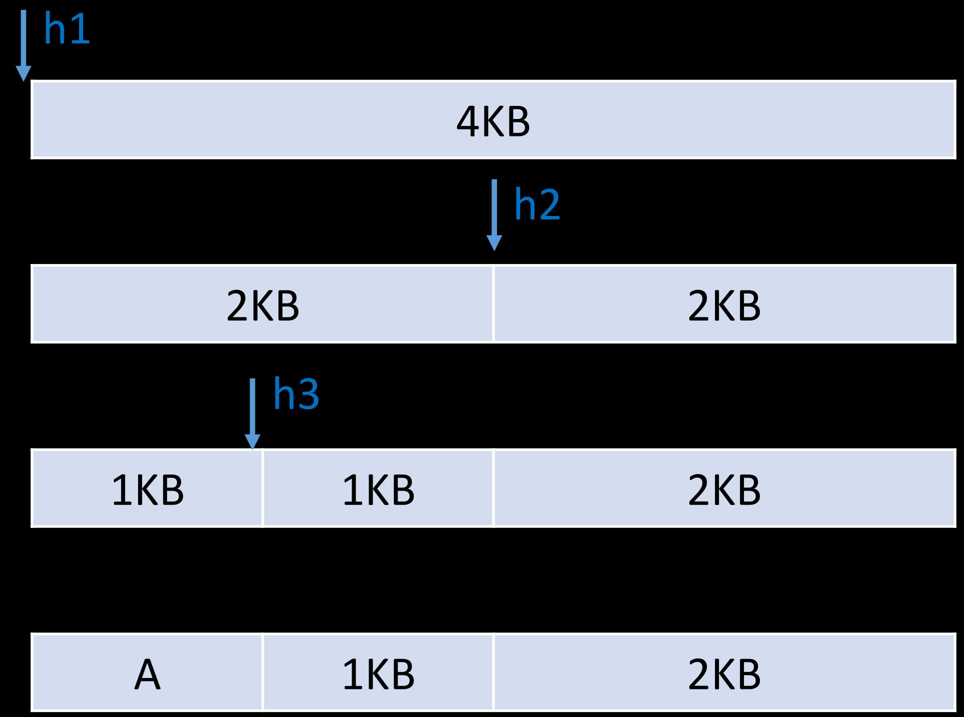
E. The design maintains a single free list containing all free chunks.

Q3 Buddy system

malloc (e.g., A=malloc(0.8KB))

1. round to powers of 2 (0.8KB -> 1KB)
2. find the non-empty free list with closest chunk size
    1. found 4KB list
3. Recursively split until having right size
4. Allocate

Assume the heap is 4KB, and initially all free

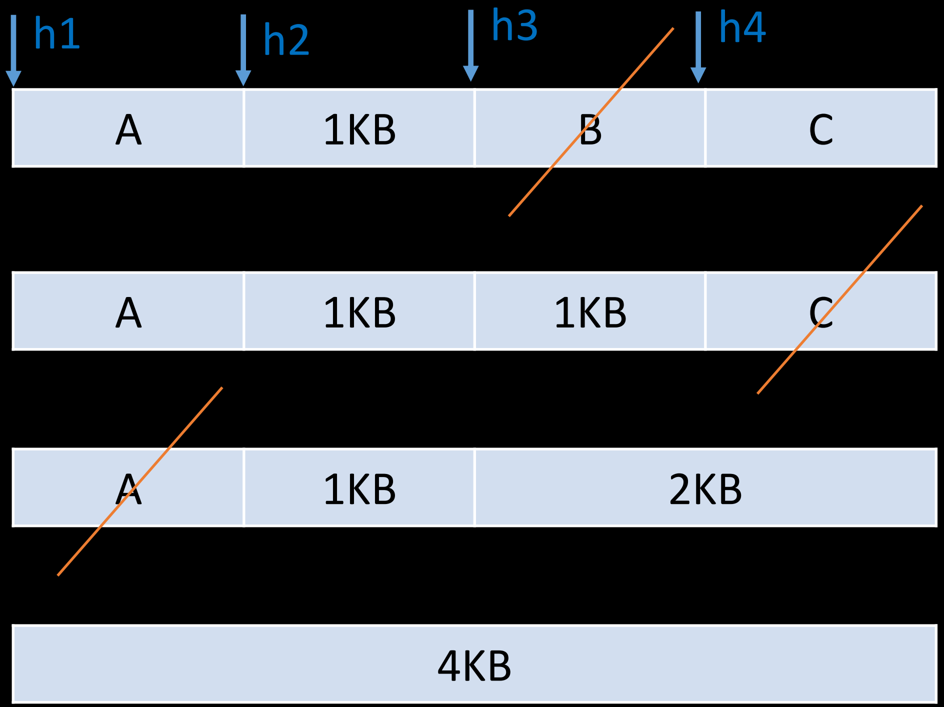


Free Lists

4KB	2KB	1KB	...
->h1	NULL	NULL	NULL
NULL	->h1->h2	NULL	NULL
NULL	->h2	->h1->h3	NULL
NULL	->h2	->h3	NULL

Q3 Buddy system

Free
1. set status bit
2. recursively coalesce with buddies



Free Lists

4KB	2KB	1KB	...
NULL	NULL	->h2	NULL
NULL	NULL	->h3->h2	NULL
NULL	->h3	->h2	NULL
h1	NULL	NULL	NULL

Lab-4

Lab4 FAQ

- Check if h is the last chunk
- Check if heap is empty
- Please read https://github.com/nyu-cso-fa21/lab4/blob/master/memlib.h and https://github.com/nyu-cso-fa21/lab4/blob/master/memlib.c

next_chunk(NULL):
- If the heap is empty: return NULL;
- Else, return the first chunk;

- Tip: reuse your code
  - e.g., “get the first chunk’s address” is implemented in next_chunk, call next_chunk in other places you need

Lab4 FAQ

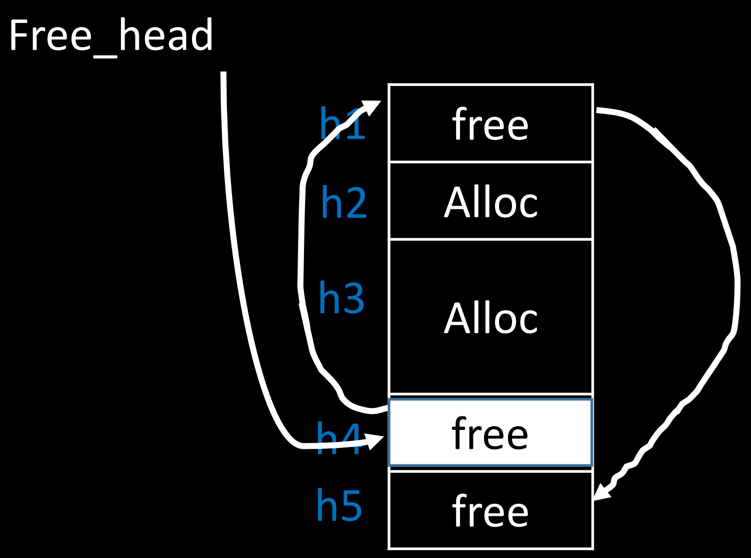
Explicit list: free

```
void free(void *p) {
    header *h = payload2header(p);
    init_free_chunk((free_hdr *)h, get_size(h));

    header *next = next_chunk(h);
    if (!get_status(next))
       h = coalesce((free_hdr *)h, (free_hdr *)next);
    header *prev = prev_chunk(h);
    if (!get_status(prev))
       h = coalesce((free_hdr *)h, (free_hdr *)prev);

    insert(&freelist, (free_hdr *)h);
}
```

(free_hdr *)h->next



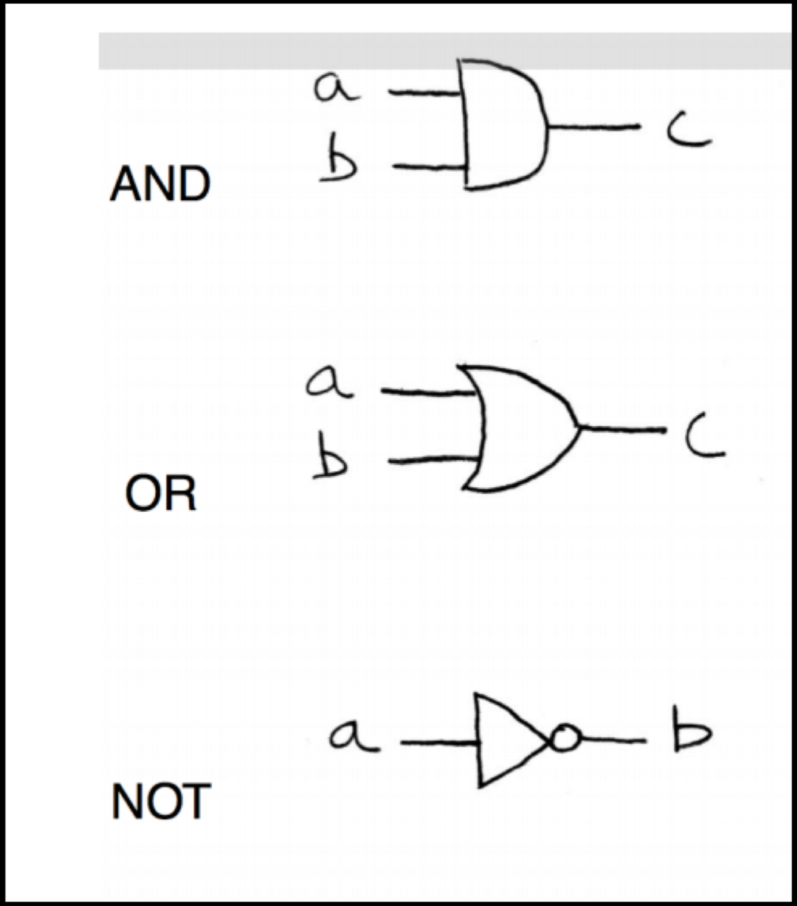
h4->next == h1, not physically consecutive!

next_chunk(h4) == h5

Combinational logic

Building Blocks

Basics



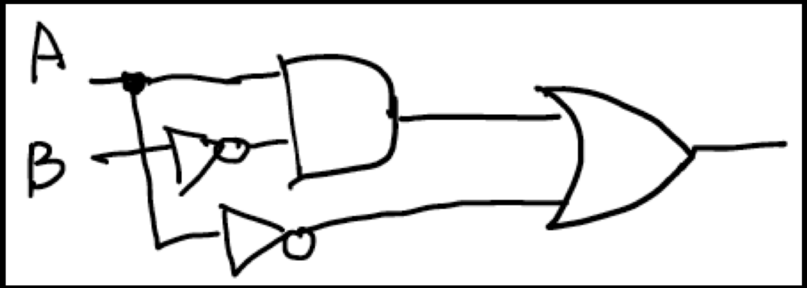
- Express (logic) functions with (logic) equations
  - A•B (AND), A+B (OR), Ā (NOT)
- Laws of Boolean algebra:
  - Basic: $A+0=A$ $A+1=1$ $A\cdot 0=0$ $A\cdot 1=A$
  - Inverse: $A+\bar{A}=1$ $A\cdot\bar{A}=0$ $\overline{A+B}=\bar{A}\cdot\bar{B}$ $\overline{A\cdot B}=\bar{A}+\bar{B}$
  - Commutativity: $A+B=B+A$ $A\cdot B=B\cdot A$
  - Associativity: $A+(B+C)=(A+B)+C$ $A\cdot(B\cdot C)=(A\cdot B)\cdot C$
  - Distribution: $A\cdot(B+C)=A\cdot B+A\cdot C$ $A+(B\cdot C)=(A+B)\cdot(A+C)$

Boolean Algebra Exercise

- Try to simplify the following equations
- $xy+xyz$
- $x(x+y)$

Boolean functions

- Boolean function: takes in boolean inputs and return boolean output
- There are three main ways to represent boolean functions
    1. As a circuit diagram (built from gates)
    2. As a set of boolean 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

More 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
    - A NAND B means NOT(A AND B)
    - 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)

Combinational Logic Design

- Basic logic design
  - Logic circuits == Boolean functions
- Combinational Logic circuit: a type of circuit without memory
  - That is, the outputs are a function ONLY of the current inputs, not of anything in the past
- How to build a combinational logic circuit with AND, OR and NOT
  - Step1: Specify the truth table
  - Step2: Output is the sum of products

Implement XOR with Combinational Logic

out =A XOR B

A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

Step1: specify the truth table

- Enumerate every possible inputs (2^N)
- Compute the output

Implement XOR with Combinational Logic

out =A XOR B

A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

(~A)*B

A*(~B)

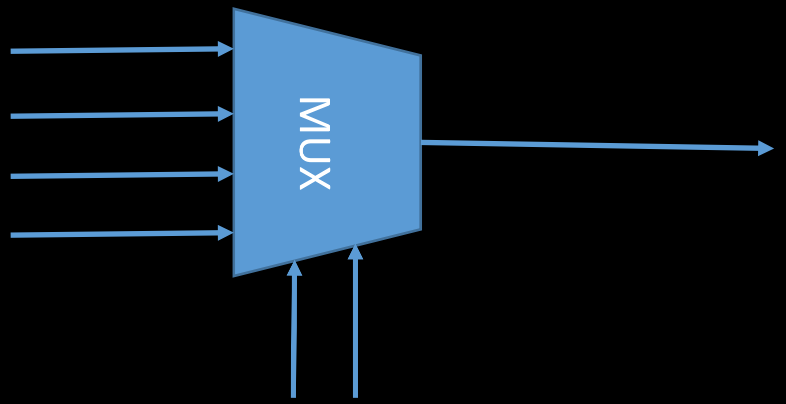
Step2: Result is the sum (OR) of products (clauses, i.e. a AND b AND …)

- Look for rows of output=1
  - write a clause for each row
    anywhere an input a is 1, write a
    Anywhere an input a is 0, write ~a
  - AND them together

- OR clauses together
- out=(~A)*B + A*(~B)

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



Multiplexor (MUX)

- 4-to-1 Multiplexor
- It can be noted that 2^N input signals require N select signals

2^N input signals

N selectors

```
Inputs        X  Y
              |  |
          |\  |  |
          |   \  |
Input A  -|00    \
Input B  -|01     |
Input C  -|10     |------Output M
Input D  -|11    /
          |   /
          |/
```