

CSC2/452 Computer Organization

Bitsets, Bitfields, Integer Arithmetic

Sreepathi Pai

URCS

September 12, 2022

Outline

Recap

Bitsets

Bitfields

Logical Operations

Integer Arithmetic

Outline

Recap

Bitsets

Bitfields

Logical Operations

Integer Arithmetic

Stuff from previous lectures that we'll need today

- ▶ Bits: 0 and 1
 - ▶ Unary functions: NOT
 - ▶ Binary functions: AND, OR, XOR, NAND, NOR, ...
 - ▶ Defined for inputs of 1 bit
 - ▶ Extensible to bit vectors and bit strings
- ▶ Machine data types
 - ▶ Byte (8 bits) smallest unit
 - ▶ Usually go up to 64 bits
- ▶ Integers
 - ▶ Sign-Magnitude, One's complement, Two's Complement, ...
- ▶ C Mapping of Types
 - ▶ machine-dependent sizes: `char`, `int`, ...
 - ▶ machine-independent sizes: `uint8_t`, `uint16_t`, ...

Outline

Recap

Bitsets

Bitfields

Logical Operations

Integer Arithmetic

Mathematical Sets

- ▶ $P = \{2, 3, 5, 7, 11, \dots\}$
- ▶ $O = \{1, 3, 5, 7, 9, 11, 13, \dots\}$
- ▶ Operations on mathematical sets:
 - ▶ Membership: $9 \in P$?
 - ▶ Union: $P \cup O$
 - ▶ Intersection: $P \cap O$
 - ▶ Complement: $\bar{O} = O' = \{2, 4, 6, 8, \dots\}$ (assuming $U = \mathbb{N}^+$)
 - ▶ Difference: $P - \{2\}$
 - ▶ Symmetric Difference: $P \oplus O = \{1, 2, 9, \dots\}$ (elements in either set but not both)

Bitsets

- ▶ Suppose you want to store *two* pieces of information ...
 - ▶ On/Off
 - ▶ Absent/Present
 - ▶ Enable/Disable
 - ▶ etc.
- ▶ about N *different* items *simultaneously*
- ▶ Example: Days of the week that I am available to meet
 - ▶ information: available/not available
 - ▶ 7 items: Sun, Mon, Tue, Wed, Thu, Fri, Sat
- ▶ How should we encode this information?
 - ▶ *not* using $\lceil \log_2(7) \rceil$ bits!
 - ▶ That's for storing 1 of N different values

Constructing Bitsets

b_6	b_5	b_4	b_3	b_2	b_1	b_0
1	0	1	1	0	0	1
SAT	FRI	THU	WED	TUE	MON	SUN

- ▶ Use 1 bit per day, total 7 bits
- ▶ If bit x is set to 1 then I'm available on day mapped to bit x
 - ▶ Interpretation matters: 1 could also be used to mean unavailable (and then 0 would be available)
 - ▶ We will use 1 for available, 0 for unavailable

Constructing Bitsets in C

```
enum DOW_BITS {  
    SUN = 1, /* bit 0 */  
    MON = 2,  
    TUE = 4,  
    WED = 8,  
    THU = 16,  
    FRI = 32,  
    SAT = 64 /* bit 6 */  
};
```

Another common option for defining constants in C:

- ▶ `#define SUN 1` (avoid for new code)

C bitwise operations

- ▶ Bitwise *NOT*(a) is $\sim a$
- ▶ Bitwise *AND*(a, b) is $a \& b$
- ▶ Bitwise *OR*(a, b) is $a | b$
- ▶ Bitwise *XOR*(a, b) is $a \wedge b$

Do not confuse *bitwise* operators with *logical/boolean* operators which are `!`, `&&`, `||`, described later in this lecture.

Marking availability

- ▶ Marking availability on Monday and Tuesday

	b_6	b_5	b_4	b_3	b_2	b_1	b_0
availability (before)	0	0	0	0	0	0	0
mark	0	0	0	0	1	1	0
availability (after)	0	0	0	0	1	1	0

Marking availability (in code)

- ▶ Marking availability on Monday and Tuesday

	b_6	b_5	b_4	b_3	b_2	b_1	b_0
availability (before)	0	0	0	0	0	0	0
mark	0	0	0	0	1	1	0
availability (after)	0	0	0	0	1	1	0

- ▶ In C code:

```
uint8_t availability = 0;  
availability = MON | TUE;
```

Adding availability

- ▶ Add availability on Monday and Tuesday

	b_6	b_5	b_4	b_3	b_2	b_1	b_0
availability (before)	1	0	1	0	1	0	0
mark	0	0	0	0	1	1	0
availability (after)	1	0	1	0	1	1	0

- ▶ What needs to change in C code?

```
availability = MON | TUE;
```

Adding availability (answer)

- ▶ Add availability on Monday and Tuesday

	b_6	b_5	b_4	b_3	b_2	b_1	b_0
availability (before)	1	0	1	0	1	0	0
mark	0	0	0	0	1	1	0
availability (after)	1	0	1	0	1	1	0

- ▶ C code

```
availability = availability | MON | TUE;
```

- ▶ Which set operation does OR resemble?

OR

- ▶ Can be used to set bits to 1
- ▶ Does not change a bit if it is already 1
- ▶ Resembles the set union operation \cup

Handling “except” adds

- ▶ All days except Monday or Tuesday

	b_6	b_5	b_4	b_3	b_2	b_1	b_0
availability (before)	0	0	0	0	0	0	0
except	0	0	0	0	1	1	0
availability (after)	1	1	1	1	0	0	1

- ▶ C code

```
uint8_t except;
```

```
except = MON | TUE;
```

```
availability = /* complete this */
```

- ▶ post-lecture answer: \sim except

Checking availability

- ▶ Am I available on Thursday?

	b_6	b_5	b_4	b_3	b_2	b_1	b_0
availability	0	0	1	0	1	1	0
check	0	0	1	0	0	0	0
result	0	0	1	0	0	0	0

- ▶ C code (what is OP?)

```
uint8_t result;  
  
result = availability OP THU;  
  
/* value of result will be 0 or THU, depending  
   on whether bit THU is zero or one respectively */
```

- ▶ answer: &

Comparing availability

- ▶ Which days are both Prof. #1 and Prof. #2 available?

	b_6	b_5	b_4	b_3	b_2	b_1	b_0
availability (Prof. #1)	1	0	1	0	1	0	0
availability (Prof. #2)	1	1	0	0	1	0	0
result	1	0	0	0	1	0	0

- ▶ C code (what is OP?)

```
uint8_t avail_prof_1 = SAT | THU | TUE;  
uint8_t avail_prof_2 = SAT | FRI | TUE;  
uint8_t result_common;  
  
result_common = avail_prof_1 OP avail_prof_2;
```

- ▶ OP: &

Enumerating Bits

- ▶ Printing out the days both are available

```
char *days[] = {"SUN", "MON", "TUE", "WED",  
                "THU", "FRI", "SAT"};  
  
int bitvalue = 1;  
int bit;  
  
for(bit = 0; bit < 7; bit++) {  
    if((result_common & bitvalue) == bitvalue) {  
        printf("both available on %s\n", days[bit]);  
    }  
  
    bitvalue = bitvalue * 2;  
}
```

Mark unavailable on Tuesday and Thursday

- ▶ Change to unavailable for Tuesday and Thursday

	b_6	b_5	b_4	b_3	b_2	b_1	b_0
availability (before)	0	0	0	0	1	1	0
remove	0	0	1	0	1	0	0
mask	1	1	0	1	0	1	1
availability (after)	0	0	0	0	0	1	0

- ▶ C code (what are OP1 and OP2)?

```
uint8_t remove = TUE | THU;
```

```
availability = availability OP1 OP2(remove);
```

- ▶ OP1: &
- ▶ OP2: ~

AND

- ▶ Can be used to set bits to 0 by masking them out
- ▶ Can test if bits are set to 1
- ▶ Resembles the membership operation \in with single bits
- ▶ But more generally, the intersection operation \cap on bitsets

Bitset Difference

Which days is Prof. #1 available, but Prof. #2 is not?

	b_6	b_5	b_4	b_3	b_2	b_1	b_0
availability (Prof. #1)	1	0	1	0	1	0	0
availability (Prof. #2)	1	1	0	0	1	0	0
difference	0	0	1	0	0	0	0

- ▶ C code (what are OP1 and OP2?)

```
uint8_t result_diff;
```

```
result_diff = avail_prof_1 OP1 OP2(avail_prof_2);
```

Bitset Difference Solution

Which days is Prof. #1 available, but Prof. #2 is not?

	b_6	b_5	b_4	b_3	b_2	b_1	b_0
availability (Prof. #1)	1	0	1	0	1	0	0
availability (Prof. #2)	1	1	0	0	1	0	0
mask	0	0	1	1	0	1	1
difference	0	0	1	0	0	0	0

► C code (OP1= $\&$ and OP2= \sim)

```
uint8_t result_diff;
```

```
result_diff = avail_prof_1 OP1 OP2(avail_prof_2);
```

Symmetric Difference

On which days is only one Prof. available?

	b_6	b_5	b_4	b_3	b_2	b_1	b_0
availability (Prof. #1)	1	0	1	0	1	0	0
availability (Prof. #2)	1	1	0	0	1	0	0
symmetric difference	0	1	1	0	0	0	0

- ▶ C code (what is OP?)

```
result_diff = avail_prof_1 OP avail_prof_2;
```

- ▶ OP: ^

XOR

- ▶ Note that $A \oplus B \oplus A = B$
- ▶ XOR is a reversible operation
- ▶ XOR is symmetric difference on bitsets, hence also represented by \oplus

Summary of bitsets

- ▶ Bitsets equivalent to sets
 - ▶ But can only store 0/1 about N items
 - ▶ Requires N bits
- ▶ Very compact and easily to manipulate if $N < 64$ (i.e. machine word size)
 - ▶ But it is easy to build bitsets of any size (just use arrays of `uint8_t` or a larger type)
- ▶ Operation mapping
 - ▶ Union: OR (`|`)
 - ▶ Intersection: AND (`&`)
 - ▶ Complement: NOT (`~`)
 - ▶ Difference: AND NOT (`& ~`)
 - ▶ Symmetric Difference: XOR (`^`)

Outline

Recap

Bitsets

Bitfields

Logical Operations

Integer Arithmetic

Bitpacking

- ▶ Sometimes space is at a premium
- ▶ Want to use as few bits as possible
- ▶ Bitfields:
 - ▶ Partition a machine word into distinct fields

Example

- ▶ Suppose we want to store day and day of the week using as little space as possible

```
uint8_t day = 9;  
uint8_t dow = MON;
```

- ▶ Day takes value 1–31
 - ▶ Bits required: ?
- ▶ Day of week takes 0 (SUN)– 6(SAT)
 - ▶ Note here we want to store 1 of 7 values
 - ▶ No need for a bitset
 - ▶ Bits required: ?
- ▶ Total storage used by two `uint8_t` variables: 16 bits
- ▶ Bits wasted: ?

Minimum Bits using Bitfields

```
enum days_of_week {  
    SUN = 0,  
    MON = 1,  
    TUE = 2,  
    WED = 3,  
    THU = 4,  
    FRI = 5,  
    SAT = 6  
};
```

	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
Mon 9	0	1	0	0	1	0	0	1
	Day of Month				Day of Week			

Constructing a bitfield

```
uint8_t daydow;  
daydow = (9 << 3) | MON;
```

- ▶ Here, 9 is the day
- ▶ It is *left-shifted* by 3 bits using the left-shift (<<) operator
 - ▶ Empty positions at right are filled with zeroes
 - ▶ Bits at left are *discarded*
 - ▶ Like multiplying by 10^3 in the metric system, except here we're multiplying by 2^3
 - ▶ 0x9 (binary 1001) becomes 0x48 (binary 0100 1000)
- ▶ Then we OR the day of the week into the freshly created lower 3 zero bits

Getting the Day of the Week

	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
Mon 9	0	1	0	0	1	0	0	1
	Day of Month				Day of Week			
result	0	0	0	0	0	0	0	1

- ▶ We want to force the day (of month) field to zero.
- ▶ What are OP and MASK?

dow = daydow OP MASK;

Getting the Day of the Week (Solution)

	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
Mon 9	0	1	0	0	1	0	0	1
	Day of Month					Day of Week		
mask	0	0	0	0	0	1	1	1
result	0	0	0	0	0	0	0	1

- ▶ We want to force the day (of month) field to zero.
`dow = daydow & 0x7;`

Getting Day of Month

	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
Mon 9	0	1	0	0	1	0	0	1
	Day of Month				Day of Week			
mask	1	1	1	1	1	0	0	0
result	0	1	0	0	1	0	0	0

- ▶ C code, all bits except lower 3
`day = daydow & (0x1f << 3);`
- ▶ Is the result what we want?

Undoing the left shift

```
day = (daydow & (0x1f << 3)) >> 3;
```

- ▶ We got 0x48 because we had shifted it left
- ▶ We can undo it by doing a right shift by 3 bits using the right-shift >>) operator
 - ▶ Bits at right are *discarded*
 - ▶ Like dividing by 2^3 , and throwing away the remainder/fractional part
 - ▶ 0x48 (binary 0100 1000) becomes 0x9 (binary 0000 1001)

Even shorter ...

```
day = (daydow >> 3) & 0x1f;
```

- ▶ Since we're using `uint8_t`, the masking is superfluous
- ▶ For unsigned integers, right shifting will fill in bits at left with 0
- ▶ Since all bits to the left of the Day of Month field are zero, we can eliminate the mask and the AND
 - ▶ But recommend always using a mask, as good programming practice

Real-life bitfields and bitsets: Unix file permissions

- ▶ Basic File permissions (can be simultaneously enabled)
 - ▶ READ
 - ▶ WRITE
 - ▶ EXECUTE
- ▶ Permissions for
 - ▶ User/Owner
 - ▶ User's Group
 - ▶ Others

	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
<code>rwxr-x---</code>	1	1	1	1	0	1	0	0	0
	User/Owner			Group			Others		

Outline

Recap

Bitsets

Bitfields

Logical Operations

Integer Arithmetic

Bitwise Operations

- ▶ Bitwise operations operate on integers and produce integers
- ▶ Logical operations operate on booleans/integers and produce booleans
- ▶ Logical operators:
 - ▶ AND: `&&`
 - ▶ OR: `||`
 - ▶ NOT: `!`
- ▶ Boolean values are TRUE and FALSE
 - ▶ In C, 0 is FALSE
 - ▶ Everything else is TRUE

Comparing Bitwise and Boolean: Results

▶ AND

- ▶ Bitwise: $3 \& 7 = 3$
- ▶ Boolean: $3 \&\& 7 = 1$

▶ OR

- ▶ Bitwise: $3 | 7 = 7$
- ▶ Boolean: $3 || 7 = 1$

▶ NOT

- ▶ Bitwise: $\sim 0 = 0\text{xff}\dots$
- ▶ Boolean: $!0 = 1$
- ▶ Boolean: $!5 = 0$

Boolean Operators: Short-circuit Behaviour

- ▶ AND
 - ▶ Boolean: $0 \ \&\& \ (5 / 0) = 0$
- ▶ OR
 - ▶ Boolean: $3 \ || \ (5 / 0) = 1$
- ▶ None of the divisions by zero in the expressions above will be executed
 - ▶ `&&` stops evaluating as soon as a subexpression evaluates to FALSE
 - ▶ `||` stops evaluating as soon as a subexpression evaluates to TRUE

Outline

Recap

Bitsets

Bitfields

Logical Operations

Integer Arithmetic

Integer format

- ▶ For `int32_t` type:
 - ▶ 1 bit sign
 - ▶ 31 bits value
 - ▶ Essentially a bitfield!

Adding two integers of different widths

```
int8_t x = -5;  
int16_t y = 10;  
  
y = y + x;
```

Shifting a signed integer to the right

```
int8_t x = -72;
```

```
x = x >> 3;
```

Unsigned integer overflow and underflow

```
/* overflow */  
uint8_t x = 255;  
x = x + 1;
```

```
/* underflow */  
uint8_t x = 0;  
x = x - 1;
```

Signed integer overflow

```
/* overflow */  
int8_t x = INT8_MAX;  
  
x = x + 1;  
  
/* underflow */  
int8_t x = INT8_MIN;  
x = x - 1;
```

Summary

- ▶ Bitsets
 - ▶ and set-like operations using bitwise operators
- ▶ Bitfields
 - ▶ shifts and masks
- ▶ Logical operators
 - ▶ differences from bitwise and short-circuit behaviour
- ▶ Integer arithmetic
 - ▶ Some things to ponder about