Lecture 2

<2016-03-30 Wed>

Contents

1 Integer

1.1 Addition

unsigned addition implements modular arithmetic

- Uadd(u, v) = (u + v) % 2^w

Tadd and Uadd has identical bit-level operation

```
int a, b;
int Usum, Tsum;
Usum = (int) ((unsigned) a + (unsigned) b);
Tsum = a + b;
```

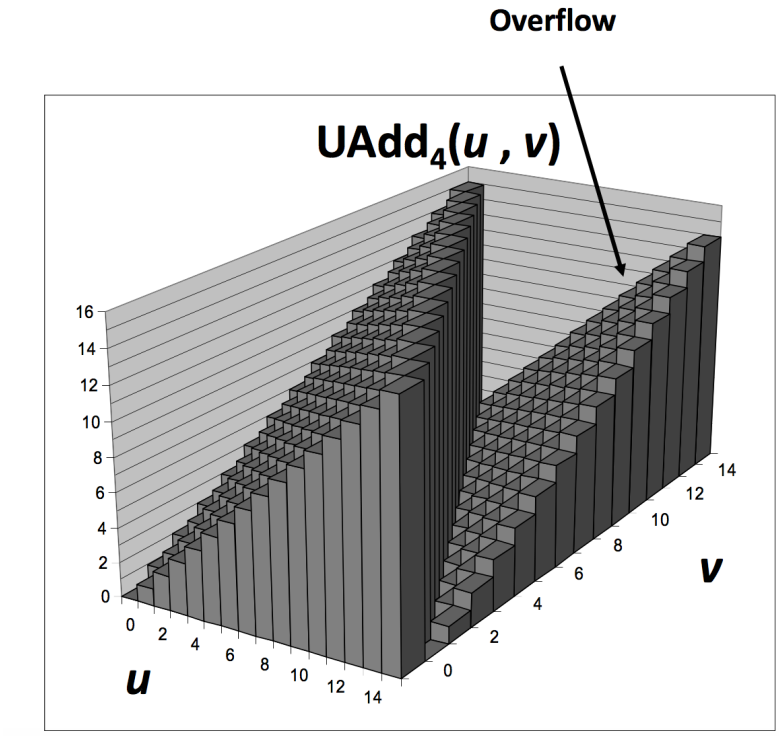

Figure 1: unsigned addition overflow

1.2 Multiplication

ignores higher order w bits

unsigned multiplication implements modular arithmetic

- Umult(u, v) = (u * v) % 2^w

1.2.1 power of 2 multiply with shift

```
(u « 5) - (u « 3) == u * 24
```

- most machines shift and add faster than multiply
  - use this method for static code, i.e. compile-time constants

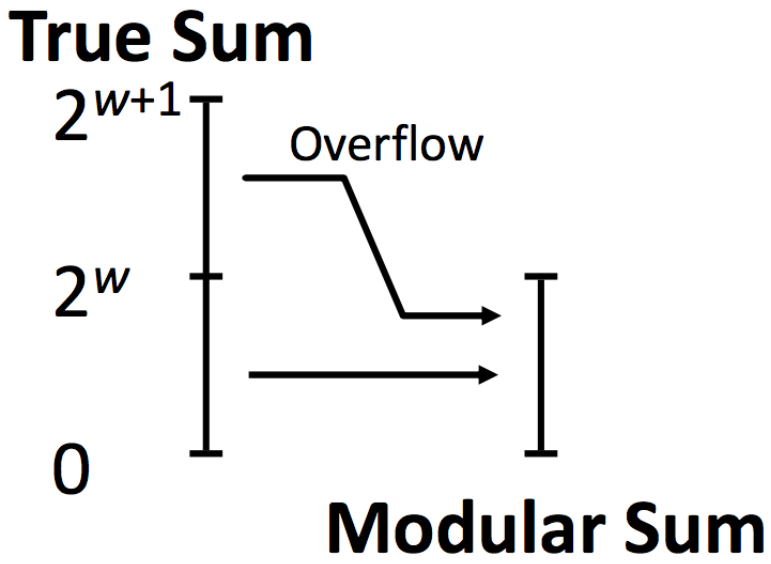

Figure 2: unsigned addition overflow

1.2.2 power of 2 unsigned division with shift

- use logical shift: pad with 0

`u >> k` == floor(u / 2^{k})

operation	decimal	hex	bin
x	15213U	`3B 6D`	`00111011 01101101`
x » 1	7606U	`1D B6`	`00011101 10110110`
y	240U	`F0`	`11110000`
y » 2	60U	`3C`	`00111100`

2 Byte-Oriented Memery Organization

word size

- nominal size of integer value and address

2.1 Address specify byte location

- Address of first byte in word

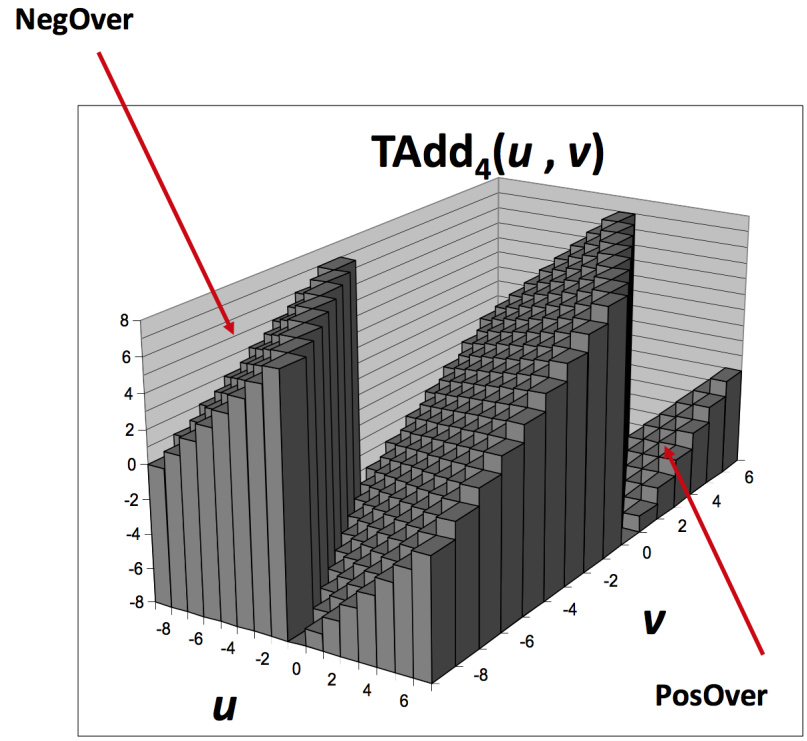

Figure 3: signed addition overflow

- Address of successive words differ depending on word size

2.2 Byte Ordering

Endianness

- Big Endian: least significant bit has highest address
- Little Endian: least significant bit has lowest address

address	0x100	0x101	0x102	0x103
big endian	01	23	45	67
little endian	67	45	23	01

```
printf("%p", address);
printf("%x", hexadecimal);
```

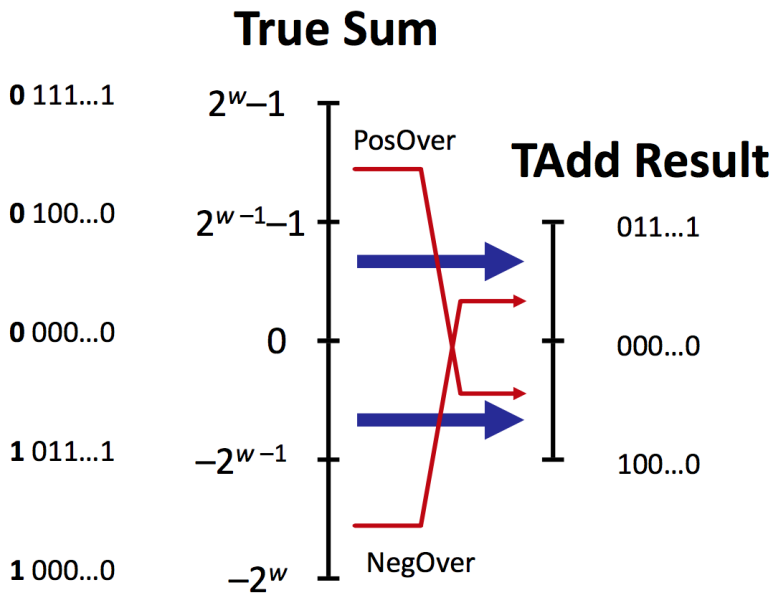

Figure 4: unsigned addition overflow

2.2.1 String

- array of characters
- byte ordering not an issue
- high addresses, latter elements(character)

2.2.2 example

```
int ix;
unsigned ux;
```

Table 1: Integer C puzzles

condition	statement	result	note
`ix & 7 == 7`	`(ix << 30) < 0`	true	last 3 bits are `111`
`ux >= 0`		true	
`ix * ix >= 0`		false	
`ix >= 0`	`-ix <= 0`	true	
`ix <= 0`	`-ix >= 0`	false	`abs(TMIN) > abs(TMAX)`
`(ix | -ix) >> 31 == -1`		true	
`ux >> 3 == ux / 8`		true	
`ix >> 3 == ix / 8`		false	signed division truncates towards 0
`x & (x - 1) != 0`		false	x == 0