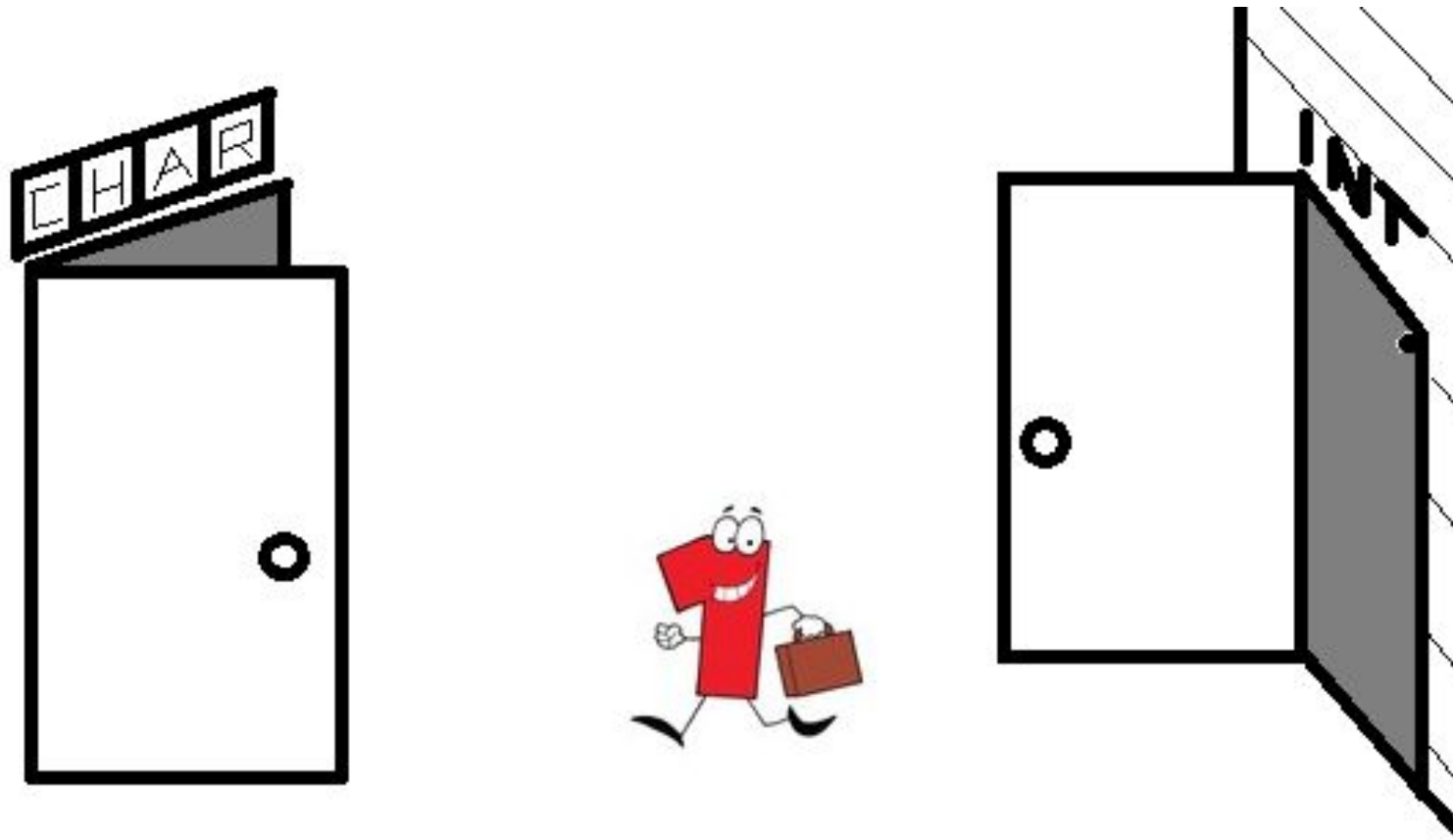# Integer Casting in C

Computer Engineering 1

CT Team: A. Gieriet, J. Gruber, R. Gübeli, M. Meli, M. Rosenthal,
A. Rüst, J. Scheier, M. Thaler

Motivation

http://www.instructables.com

Agenda

- Type conversion
  - signed ↔ unsigned
  - Extension
  - Truncation

Learning Objectives

At the end of this lesson you will be able

- to explain the casting of integer types in C

- to apply the assembly instructions associated with casting

- to say how a given memory content is interpreted for different integer types in C

- to give the memory content after storing different C integer types

Integer Casting in C

- **Integer ranges based on word sizes**

8-bit	hex	unsigned	signed
	0x00	0	0
	...	...	...
	0x7F	127	127
	0x80	128	−128
	...	...	...
	0xFF	255	−1

16-bit	hex	unsigned	signed
	0x0000	0	0
	...	...	...
	0x7FFF	32'767	32'767
	0x8000	32'768	−32'768
	...	...	...
	0xFFFF	65'535	−1

32-bit	hex	unsigned	signed
	0x0000 0000	0	0
	...	...	...
	0x7FFF'FFFF	2'147'483'647	2'147'483'647
	0x8000'0000	2'147'483'648	−2'147'483'648
	...	...	...
	0xFFFF'FFFF	4'294'967'295	−1

Integer Casting in C: Type conversion

- **signed ←→ unsigned**

`int8_t`	←→	`uint8_t`
`int16_t`	←→	`uint16_t`
`int32_t`	←→	`uint32_t`
`int64_t`	←→	`uint64_t`

- **Extension**

`int_8`	→	`int16_t`	→	`int32_t`	→	`int64_t`	signed
`uint_8`	→	`uint16_t`	→	`uint32_t`	→	`uint64_t`	unsigned

- **Truncation**

`int64_t`	→	`int32_t`	→	`int16_t`	→	`int_8`	signed
`uint64_t`	→	`uint32_t`	→	`uint16_t`	→	`uint_8`	unsigned

Integer Casting in C

- **signed ←→ unsigned**

signed $\quad -b_3 \cdot 2^3 + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0$

unsigned $\quad +b_3 \cdot 2^3 + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0$

Casts in **red** area

➔ Small negative numbers turn into large positive numbers

➔ Large positive numbers turn into small negative numbers

binary	unsigned	signed 2' compl.
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

Integer Casting in C

- **Casting**   unsigned   →   signed

    signed   →   unsigned

| Bit representation stays the same, but interpretation changes |

- **Example 4-Bit**
  - 1011b   → Interpretation as unsigned   11d

    → Interpretation as signed   -5d

Integer Casting in C

- **Example 1: signed 8-bit → unsigned 8-bit**
  - Bit representation stays the same, interpretation changes

```
dec    hex    EK     ZK = EK + 1
10d    0x0A   0xF5   0xF6
```
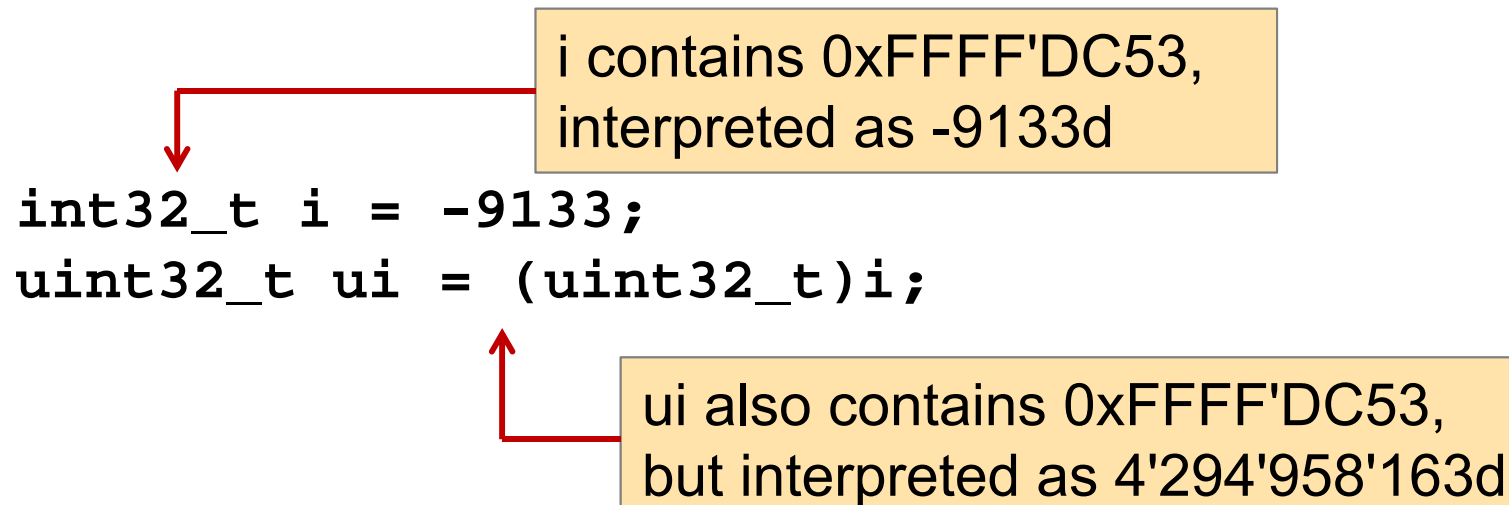
c contains 0xF6,
interpreted as -10d

```
int8_t c = -10;
uint8_t uc = (uint8_t)c;
```

uc also contains 0xF6,
but interpreted as 246d

Integer Casting in C

- **Example 2: signed 32-bit → unsigned 32-bit**
  - Bit representation stays the same, interpretation changes

dec	hex	EK	ZK = EK + 1
9133d	0x0000'23AD	0xFFFF'DC52	0xFFFF'DC53

i contains 0xFFFF'DC53, interpreted as -9133d

```
int32_t i = -9133;
uint32_t ui = (uint32_t)i;
```

ui also contains 0xFFFF'DC53, but interpreted as 4'294'958'163d

Explicit cast is not even required
```
uint32_t ui2 = i;  // implicit cast
```

Integer Casting in C

- **Example 3: Cast unsigned 32-bit → signed 32-bit**
  - Bit representation stays the same, interpretation changes

dec	hex
4'294'964'632d	0xFFFF'F598

uix contains 0xFFFF'F598,
interpreted as 4'294'964'632

```
uint32_t uix = 4294964632;
int32_t ix = uix;    // implicit cast
```
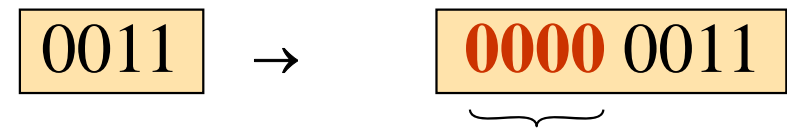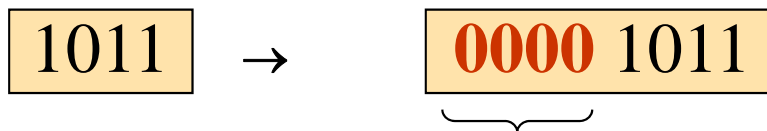
ix also contains 0xFFFF'F598,
but interpreted as -2664d

Integer Casting in C

- **If one of the operands is unsigned, C performs an implicit cast for the signed values**

  - Example n = 32:   signed ∈ [-2'147'483'648 , 2'147'483'647]
  - Can lead to strange results (red lines)

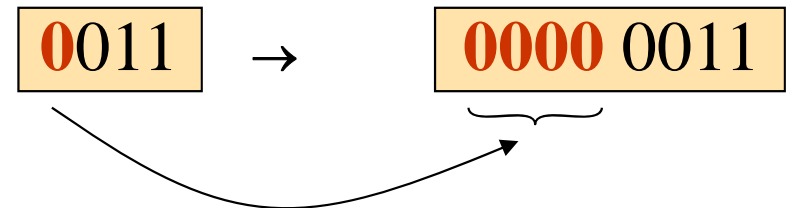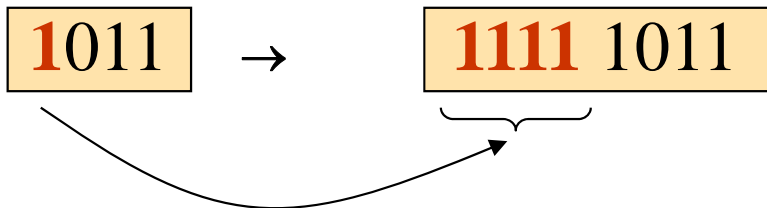| Expression	Type	Evaluation
0 == 0U	unsigned	1
-1 < 0	signed	1
-1 < 0U	unsigned	0
2'147'483'647 > -2'147'483'647 - 1	signed	1
2'147'483'647U > -2'147'483'647 – 1	unsigned	0
2'147'483'647 > (int) 2'147'483'648U	signed	1
-1 > -2	signed	1
(unsigned) -1 > -2	unsigned	1

Examples: R. Bryant, D. O'Hallaron

Integer Casting

- **Extension: 4 Bit → 8 Bit**
  - **Unsigned → Zero Extension**

    | 1011 | → | **0000** 1011 | | 0011 | → | **0000** 0011 |

  - **Signed → Sign Extension**

    | **1**011 | → | **1111** 1011 | | **0**011 | → | **0000** 0011 |

Integer Casting: Sign Extension

- **Sign Extension Cortex-M4 (signed values)**
  - Extend word-length without changing value
  - SXTB   Extends an 8-bit value to a 32-bit value
  - SXTH   Extends a 16-bit value to a 32-bit value

- **Zero Extension Cortex-M4 (unsigned values)**
  - Extend word-length, fill with zeroes
  - UXTB   Extends an 8-bit value to a 32-bit value
  - UXTH   Extends a 16-bit value to a 32-bit value

Examples

```
SXTB  R3, R10  ; Extract lowest byte of the value in R10,
               ; sign extend it and write the result to R3
UXTH  R2, R3   ; Extract lower two bytes of the value in R3,
               ; zero extend it and write the result to R3
```

Integer Casting: Sign Extension

- **Example Sign Extension**

```
int16_t sx =  15213;
int32_t ix = (int32_t)sx;


int16_t sy = -15213;
int32_t iy = (int32_t)sy;
```

	dec	hex	bin
sx	15213	0x3B6D	00111011 01101101
ix	15213	0x0000'3B6D	00000000 00000000 00111011 01101101
sy	-15213	0xC493	11000100 10010011
iy	-15213	0xFFFF'C493	11111111 11111111 11000100 10010011

signed Integer Types: from small to large

→ **Sign bit is copied to the left**

Integer Casting: Truncation

- **Truncation: Reduce number of digits**
  - Cast cuts the left most digits
- **Unsigned** → modulo Operation

```
uint32_t x = 287962;
uint16_t sx = (uint16_t)x;
uint32_t y = (uint32_t)sx;
```

```
0x000464DA →   287'962
    0x64DA →    25'818
0x000064DA →    25'818
```

- **Signed** → possible change of sign!

```
int32_t x  = 53191;
int16_t sx = (int16_t) x;
int32_t y  = (int32_t) sx;
```

```
0x0000CFC7 →   53'191
    0xCFC7 →  -12'345
0xFFFFCFC7 →  -12'345
```

Conclusion

- **Integer Casts**
  - Type Conversions

- **signed – unsigned**
  - Small negative numbers correspond to large positive numbers

- **Extensions**
  - Add additional bits
    - signed   sign extension   copy sign bit to the left
    - unsigned   zero extension   fill left bits with zero

- **Truncations**
  - Cut left most digits
    - signed   possible change of sign
    - unsigned   results in modulo operation