Remaining Data Types

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

Contents

Learning Objectives

School of Engineering
InES Institute of Embedded Systems

- At the end of this lesson you will be able
  - to explain the structure and notation of real numbers in decimal and binary
  - to translate real numbers between decimal and binary
  - to explain how arrays are stored and how to access to single elements
  - to discuss how structs and strings are stored

Fractional Numbers

- **Binary representation**

2^i	2^{i-1}		4	2	1	1/2	1/4	1/8		2^{-j}
b_i	b_{i-1}	•••	b_2	b_1	b_0	b_{-1}	b_{-2}	b_{-3}	•••	b_{-j}

- **Value**

$$\sum_{k=-j}^{i} b_k \times 2^k$$

- **Examples**
  - 5 3/4 101.11_2
  - 2 7/8 10.111_2

Scalar Types: Real Numbers

School of Engineering
InES Institute of Embedded Systems

- **Real numbers: general notation**

$$\text{Value} = \text{Sign} \cdot \text{Fraction} \cdot \text{Base}^{\text{Exponent}}$$

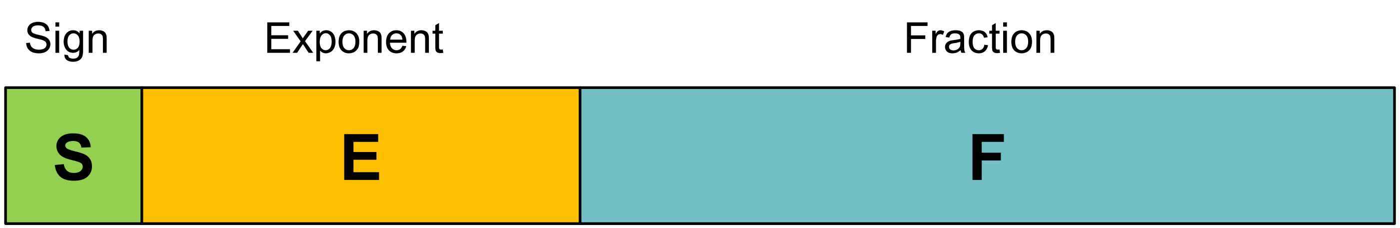
- **Normalized scientific notation**
  - Single non-zero digit to the left of the decimal (binary) point

- **Decimal vs. binary**
  - Decimal: $0.00002796 = 2.796 \cdot 10^{-5}$
  - Binary: $0.000010110111 = 1.0110111 \cdot 2^{-5}$

IEEE Standard 754 / 854

- **Defines how floating point numbers are represented**
  - Easy exchange of data between machines
  - Simplifies hardware algorithms
  - Exists since 1980

- **Binary notation**



IEEE Standard 754 / 854

Fraction

- Binary notation
- Representing normalized numbers → number of form ***1***.xxxx..
- In IEEE 754 standard, the ***1*** is implicit

$$\boxed{\text{Fraction value} = (1 + F)}$$

- Example

$$\boxed{1.265625\ d = 1.010001\ b \rightarrow F = 010001}$$

Remark: 1.010001 b = $(1 + 0 \times 2^{-1} + 1 \times 2^{-2} + \ldots + 1 \times 2^{-6})$ d

IEEE Standard 754 / 854

Exponent

- Binary excess notation

$$\boxed{\text{Exponent value} = (E - \text{Bias})}$$

- Bias: 127 (float) / 1023 (double)

- Example (float):

$$\boxed{0111'1010b = 122 \rightarrow 122 - 127 = -5 \rightarrow \text{Value} = 2^{-5}}$$

$$\boxed{\text{Value} = 2^{-5} \rightarrow -5 + 127 = 122 = 0111'1010b}$$

IEEE Standard 754 / 854

zh aw School of Engineering
InES Institute of Embedded Systems

- **Exponent Single precision (float)**

S	E	F

E		Meaning
0000 0000		Reserved
0000 0001		-126_{10}
0000 0010		-125_{10}
0111 1111		0_{10}
1111 1110		127_{10}
1111 1111		Reserved

→ Unnormalized
→ See later…

→ Not a Number
→ See later…

IEEE Standard 754 / 854

- **Sign**

  - 1 bit
  - Sign value = $(-1)^S$

IEEE Standard 754 / 854

Sign and Magnitude Representation

S	E	F

$$\text{Value} = (-1)^{S} \cdot (1 + F) \cdot 2^{(E - \text{Bias})}$$

- More exponent bits → wider range of numbers
- More fraction bits → higher precision

Exercise: Single Precision (float)

School of Engineering
InES Institute of Embedded Systems

- **Decimal 7.5 d = ?**

Exercise: Single Precision (float)

- **Binary 10111111'01010000'00000000'00000000 = ?**

Scalar Types: Single Precision

School of Engineering
InES Institute of Embedded Systems

Special Values

E	F	S	Result	
255	$\neq 0$		Not a Number (NaN)	
255	0	1	$- \infty$	
255	0	0	∞	
$0 < E < 255$			$(-1)^S \cdot (1 + F) \cdot 2^{(E\text{-Bias})}$	Normalized
0	$\neq 0$		$(-1)^S \cdot (0 + F) \cdot 2^{(1\text{-Bias})}$	Unnormalized
0	0	1	$- 0$	
0	0	0	$+ 0$	

Ranges (Single Precision)

- **Normalized (positive):** $(1 + F) \cdot 2^{(E\text{-}Bias)}$
  - Smallest `0 00000001 00000000000000000000000` $+2^{-126}(1+0) = 2^{-126}$
  - Largest `0 11111110 11111111111111111111111` $+2^{127}(1+(1-2^{-23}))$

- **Unnormalized (positive):** $(0 + F) \cdot 2^{(1\text{-}Bias)}$
  - Smallest `0 00000000 00000000000000000000001` $+2^{-126}(2^{-23}) = 2^{-149}$
  - Largest `0 00000000 11111111111111111111111` $+2^{-126}(1-2^{-23})$

IEEE Standard 754 / 854

- **Single precision (float)**
  - 32 bits
  - Exponent bias: 127
  - Dynamics: $-3.4 \cdot 10^{38}$ ... $-1.17 \cdot 10^{-38}$, 0, $1.17 \cdot 10^{-38}$ ... $3.4 \cdot 10^{38}$
  - Resolution: $2^{-24} = 0.6 \cdot 10^{-7} \rightarrow$ 7 digits

1 bit	8 bits	23 bits
S	E	F

- **Double precision (double)**
  - 64 bits
  - Exponent bias: 1023
  - Dynamics: $-1.8 \cdot 10^{308}$ ... $-2.2 \cdot 10^{-308}$, 0, $2.2 \cdot 10^{-308}$ ... $1.8 \cdot 10^{308}$
  - Resolution: $2^{-53} = 0.11 \cdot 10^{-15} \rightarrow$ 15 digits

1 bit	11 bits	52 bits
S	E	F

Precision Errors

- **Precision:**

```
int main(void) {
        float a = 1.8;
        float b = 18;
        if(a*10 == b) printf("\nNumbers are the same\n\n");
        else          printf("\nNumbers differ!!\n\n");
}
```

  - Output: `Numbers are the same`

```
int main(void) {
        float a = 1.81;
        float b = 18.1;
        if(a*10 == b) printf("\nNumbers are the same\n\n");
        else          printf("\nNumbers differ!!\n\n");
}
```

  - Output `Numbers differ!!`

Precision Errors

Precision:

```
int main(void) {
        int x   = 33554431;
        float y = 33554431;
        printf("int: %d\n", x );
        printf("float %f\n", y );
}
```

- Output:

```
int: 33554431
float: 33554432.000000
```

- Single Precision float of **33554431**:

Binary32: $\boxed{\text{4C000000}}$

Status	Sign [1]	Exponent [8]	Significand [23]
Normal	0 (+)	10011000 (+25)	1.00000000000000000000000 (1.0)

$2^{25} = 33554432$

https://babbage.cs.qc.cuny.edu/IEEE-754

Precision Errors

- **Patriot Missile Error**

  - During the Gulf War in 1991, a U.S. Patriot missile failed to intercept an Iraqi Scud missile, and 28 Americans were killed.
  - The problem was caused by the inaccuracy of the binary representation of 0.10.
    - The Patriot incremented a counter once every 0.10 seconds.
    - It multiplied the counter value by 0.10 to compute the actual time.
  - However, the (24-bit) binary representation of 0.10 actually corresponds to 0.099999904632568359375, which is off by 0.000000095367431640625.
  - The system was running 100 hours, the time ended up being off by 0.34 seconds - enough time for a Scud to travel 500 meters!

Arrays in C

- **Pointer to first element of array**

```
int main(void) {
    int j, sum = 0;
    int array[5];
    int *ptr;                          // pointer to integer
    for (j = 0; j < 5; j++) {
        array[j] = j;
    }
    ptr = array;                       // copy pointer
    for (j = 0; j < 5; j++) {
        sum = sum + *ptr;              // add value ptr points
        ptr++;                         // point to next element
    }
}
```

```
for (j = 0; j < 5; j++) {
    sum = sum + array[j];
}
```

Arrays in C

School of Engineering
InES Institute of Embedded Systems

- **Multidimensional arrays are "Arrays of Arrays"**

```
void main(void) {
   int array [3][3];
   int *ptr;
   int i, j, k;
   j = 0;                           // fill array with ascending
   for (i = 0; i < 3; i++) {        // numbers
      for (k = 0; k < 3; k++) {
         array [i][k] = j++;
   }  }
   ptr = array [0];                 // get pointer to first element
   for (i = 0; i < 9; i++) {
      printf("index: %d, value: %d\n", i, *ptr);
      ptr++;
   }  }
```

- Array a[3][3]: $\begin{bmatrix} a00 & a01 & a02 \\ a10 & a11 & a12 \\ a20 & a21 & a22 \end{bmatrix}$
- Is stored as: $\begin{bmatrix} a00 & a01 & a02 & a10 & a11 & a12 & a20 & a21 & a22 \end{bmatrix}$

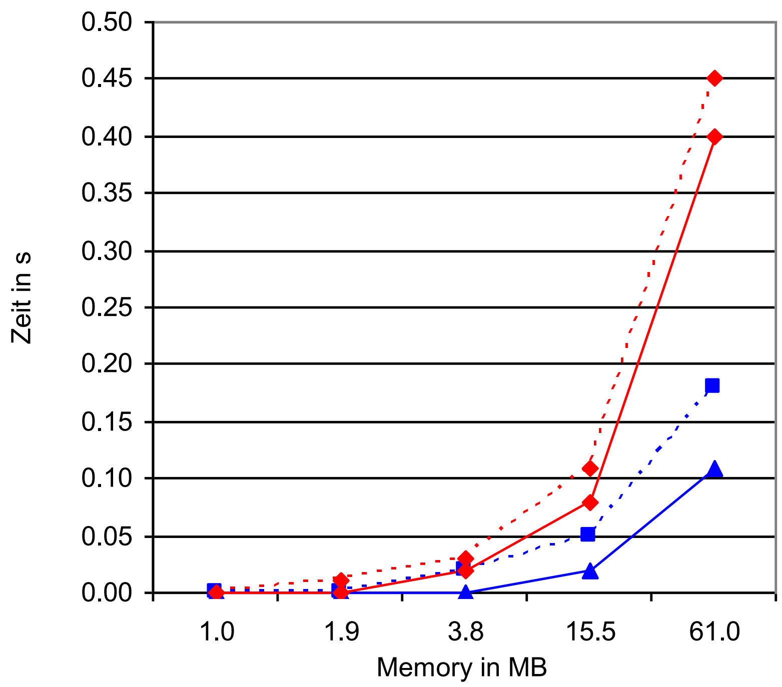
Arrays: Who Cares?

- **Access to arrays: g++, Linux**
  - long int array [N][N]
  - N = 500, 700, 1000, 2000, 4000, 8000, 12000, 16000, 20000
    M = 1, 1.9, 3.8, 15.3, 61.0, 244, 550, 980, 1500 MB
  - Blue: CPU-time

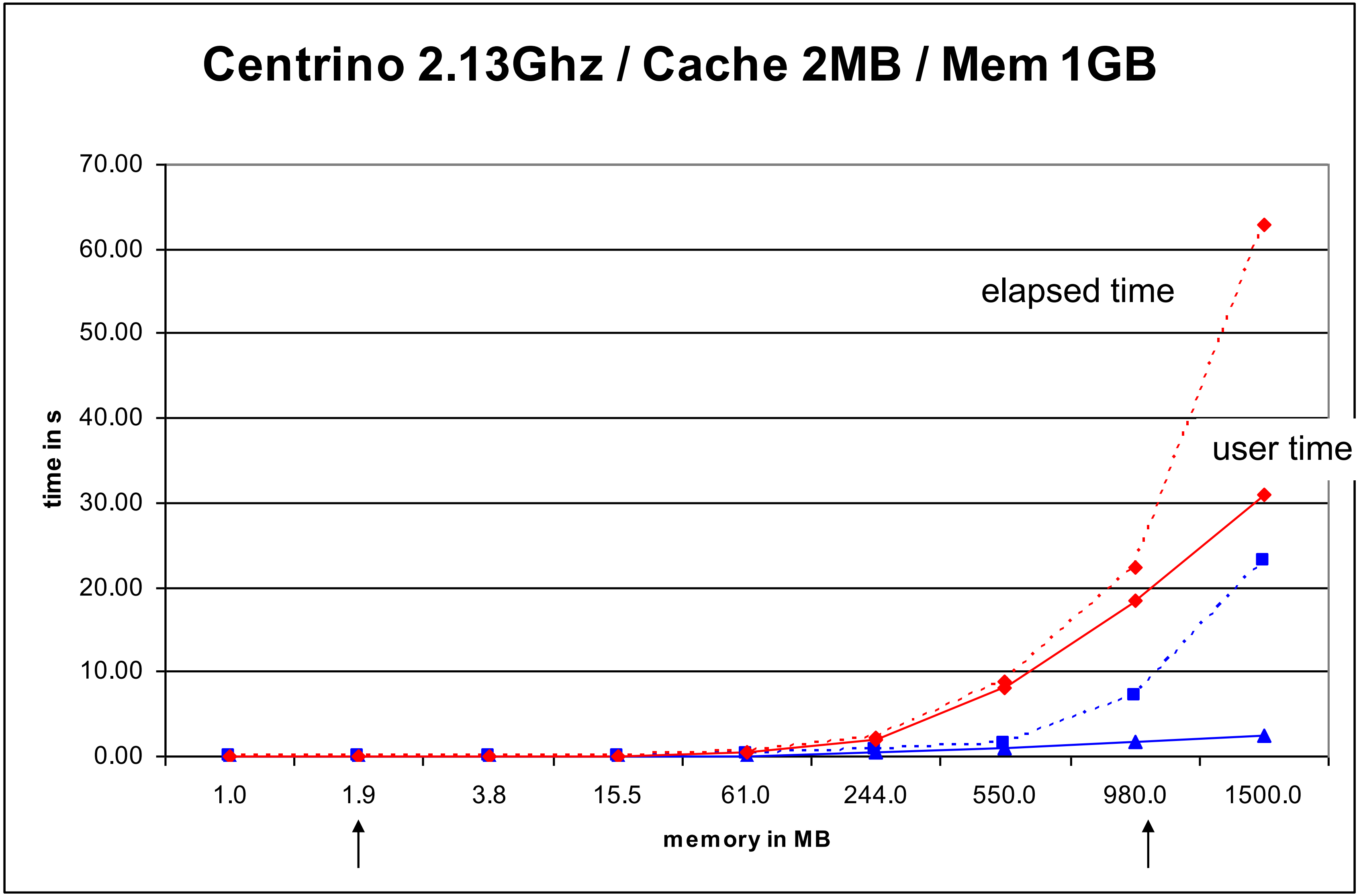
```
for (i = 0; i < s; i++) {
    for (j = 0; j < s; j++) {
        array[i][j]=i+j;
}
```

  - Red: CPU-time

```
for (i = 0; i < s; i++) {
    for (j = 0; j < s; j++) {
        array[j][i]=i+j;
}
```



... arrays



Centrino 2.13Ghz / Cache 2MB / Mem 1GB
70.00
60.00
50.00
40.00
30.00
20.00
10.00
0.00
time in s
elapsed time
user time
1.0
1.9
3.8
15.5
61.0
244.0
550.0
980.0
1500.0
memory in MB

Animation

Strings in C

School of Engineering
InES Institute of Embedded Systems

Strings in C are not objects

- Declaration with char arrays
- Example

```
char Name[16] = "Meier";
```

- Question: How is the end of a string recognised?
- All Strings in C are 0 – terminated
  → after the last character follows a '0'

'M'	'e'	'i'	'e'	'r'	'\0'	'i'	'e'	'a'	'Q'	'u'	'a'	'r'	'\0'	'l'	'l'

valid String

Terminator

undefined content

- Length of strings not stored → count characters to '\0'

Struct in C

- **Structs contain different data that belong together**
  - Declaration

```
struct person {
  char       name[8];
  char       prename[8];
  int        pers_nr;
  unsigned   char day_of_birth;
  unsigned   char month_of_birth;
};
```

  - Definition

```
struct person aperson;
```

  - Usage

```
strcpy(aperson.Name, "Meier");
strcpy(aperson.prename, "Peter");
```

Struct Memory Layout

- **Access to struct**

```
struct person {
  char      name[8];
  char      prename[8];
  int       pers_nr;
  unsigned  char day_of_birth;
  unsigned  char month_of_birth;
};
```

  - Using base address

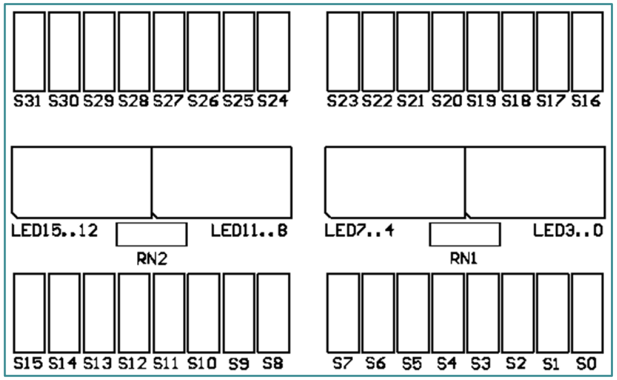
- name: base address + 0
- prename: base address + 8
- pers_nr: base address + 16
- day_of_birth: base address + 18
- month_of_birth: base address + 19

```
sizeof(struct person) = 20
```

Identifier	Memory Content	Address	
name	M	0x0000'2000	← Base Address
	e	0x0000'2001	
	i	0x0000'2002	
	e	0x0000'2003	
	r	0x0000'2004	
	\0	0x0000'2005	
		0x0000'2006	
		0x0000'2007	
prename	P	0x0000'2008	
	e	0x0000'2009	
	t	0x0000'200A	
	e	0x0000'200B	
	r	0x0000'200C	
	\0	0x0000'200D	
		0x0000'200E	
		0x0000'200F	
pers_ne	156	0x0000'2010	
	034	0x0000'2011	
day_of_birth	15	0x0000'2012	
month_of_birth	8	0x0000'2013	

Union Memory Layout

School of Engineering
InES Institute of Embedded Systems



Memory Map

S 7	S 6	S 5	S 4	S 3	S 2	S 1	S 0	0x6000'0200
S 15	S 14	S 13	S 12	S 11	S 10	S 9	S 8	0x6000'0201
S 23	S 22	S 21	S 20	S 19	S 18	S 17	S 16	0x6000'0202
S 31	S 30	S 29	S 28	S 27	S 26	S 25	S 24	0x6000'0203

```
typedef union {
    struct {
        volatile uint8_t S7_0;
        volatile uint8_t S15_8;
        volatile uint8_t S23_16;
        volatile uint8_t S31_24;
    } BYTE;
    struct {
        volatile uint16_t S15_0;
        volatile uint16_t S31_16;
    } HWORD;
    volatile uint32_t WORD;
} reg_ct_dipsw_t;

#define CT_DIPSW ((reg_ct_dipsw_t *) 0x60000200)
...
sizeof (reg_ct_dipsw_t) = 4
```

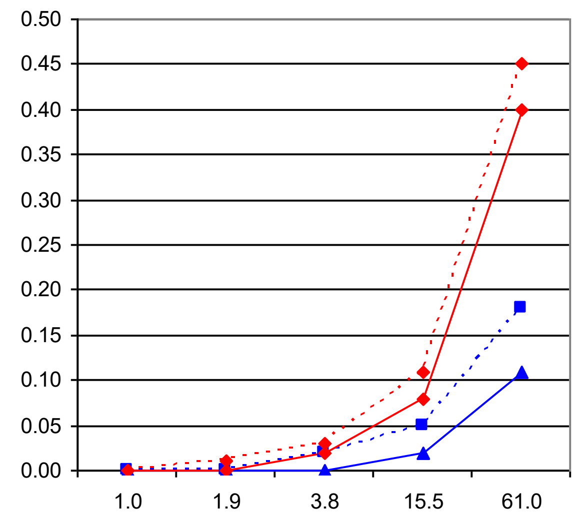
WORD	HWORD	BYTE	Address
	s15_0	s7_0	0x6000'0200
		s15_8	0x6000'0201
	s31_16	s23_16	0x6000'0202
		s31_24	0x6000'0203

Conclusion

- **Real numbers**
  - IEEE Standard 754 / 854
  - Single vs. double precision

1 bit	8 bits	23 bits
S	E	F

- **Memory Layout of**
  - Arrays
  - Strings
  - Structs



'M'	'e'	'i'	'e'	'r'	'\0'	'i'	'e'	'a'	'Q'	'u'	'a'	'r'	'\0'	'I'	'I'