ECOAR – Computer Architecture
Lecture notes

Module 2 – Data

Electrical and Computer Engineering
Faculty of Electronics and Information Technology
Warsaw University of Technology

HUMAN CAPITAL
HUMAN – BEST INVESTMENT!

EUROPEAN UNION
EUROPEAN
SOCIAL FUND

Project is co-financed by European Union within European Social Fund

Outline

- Data types
- Binary data representation
- Data in computer's memory
- Vector data types

Data types

- Boolean (True/False)
- Text characters
- Numbers
  - integer
    - unsigned
    - signed
  - non-integer
    - Fixed point
    - Floating point
- Sounds and other unidimensional signals
- Raster images

Binary data representation

- Computer operates solely on groups of bits (binary digits), treating them as binary numbers
  - Contemporary computers usually operate on binary words of lengths equal 8×2^n bits (8, 16, 32, 64)
- Non-numeric data must be expressed using numbers

Alphanumeric data

- Every character is represented by a number denoting its position in code table
- Most frequently used codes:
  - ASCII – 128 positions including upper- and lowercase Latin letters
  - Extended ASCII – 256 positions
    - First 128 positions identical to ASCII, the remaining 128 positions used for national characters and special symbols
    - problem: different codes are needed for various parts of the world
  - EBCDIC family of codes – 256 symbols, used mainly by IBM
  - UNICODE
    - Initially 2^{16}, currently up to 2^{21} positions
    - Currently contains approx. 150000 characters
    - Covers all alphabetic characters used in the world

ASCII

- American Standard Code for Information Interchange
- Developed for teletype devices, later used in computers
- 128 positions, 95 visible and 33 invisible
- Invisible – use codes 0..31 (0x00..0x1f) and 127 (0x7f):
  - White spaces and formatting codes (tabs, new line, new page)
  - Transmission and device control
  - CAN – cancel/delete - code 127 (0x7f)
- Visible – use codes 32..126 (0x20..0x7e):
  - space (code 32)
  - Digits
  - Latin letters
  - Punctuation marks
  - Common mathematical symbols

What a programmer should know about ASCII

- Control codes - positions 0..31, including
  - CR – carriage return – 13
  - LF – line feed – 10
  - Also important: HT, FF, BSP, BEL
- Space – 32 (0x20)
- digits 0..9 – codes 48..57 (0x30..0x39)
  - to get digit's value, subtract \`0\` (0x30) from character representing it
- Letters in alphabetic order
  - Upper case – 65..90 (0x41..0x5a)
  - Lower case – 97..122 (0x61..0x7a)
  - Distance between lower and upper case of a letter is 32 (0x20)
- Other visible chars fit in the range 33..126
- 127 – the very last special code (cancel/delete)

Extended codes based on ASCII

- 256 code points – 8-bit representation
  - First 128 positions identical to ASCII
  - Next 128 positions used to represent characters from a selected set of alphabets, ex.
    - Slavic
    - Nordic
    - Cyrillic
    - Greek
- Many code tables of different origin used in some countries
  - ISO8859 family (Polish characters present in ISO8859-2)
  - Microsoft – “code pages” - CP (Poland – CP852, CP1250)
  - Local usage (Polish - Mazovia, Polgaz)

HUMAN CAPITAL
WARSAW UNIVERSITY OF TECHNOLOGY DEVELOPMENT PROGRAMME
EUROPEAN UNION EUROPEAN SOCIAL FUND

UNICODE and UTF-8

- Unicode, introduced in late 1980s, is a 21-bit character code designed to include all alphabetic characters used in the world
- Symbolic notation of Unicode character: U+<hex_code>
- UTF-8 is the most common Unicode representation
- A character is represented by 1..4 octets
  character binary code

b_{20}	b_{19}	b_{18}	b_{17}	b_{16}	b_{15}	b_{14}	b_{13}	b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

is represented as:

- ASCII chars U+00..U+7F → 1 octet: $0b_6b_5b_4b_3b_2b_1b_0$
- U+80..U+7FF → 2 octets: $110b_{10}b_9b_8b_7b_6$ $10b_5b_4b_3b_2b_1b_0$
- U+800..U+FFFF → 3 octets:
  $1110b_{15}b_{14}b_{13}b_{12}$ $10b_{11}b_{10}b_9b_8b_7b_6$ $10b_5b_4b_3b_2b_1b_0$
- U+10000..U+10FFFF → 4 octets:
  $11110b_{20}b_{19}b_{18}$ $10b_{17}b_{16}b_{15}b_{14}b_{13}b_{12}$ $10b_{11}b_{10}b_9b_8b_7b_6$ $10b_5b_4b_3b_2b_1b_0$

UNICODE

Text strings

- Text string is a sequence of characters
- The software must know where the string ends
- Two common conventions:
  - Character string end identified by a special character
    - In C and similar languages, the string ends with NUL (zero) code
    - in other environments ‘$’ or LF was used
  - An explicit string length specified as binary number, stored before the text
    - common in Basic language

HUMAN CAPITAL HUMAN – BEST INVESTMENT! WARSAW UNIVERSITY OF TECHNOLOGY DEVELOPMENT PROGRAMME EUROPEAN UNION EUROPEAN SOCIAL FUND

Sounds and images

- Sound:
  - Voltage analog of acoustic pressure sampled with frequency dependent on the quality requirements (from 8 to 48 kHz)
  - Sample values stored as integer numbers
- Raster images
  - Represented as rectangular arrays of square picture elements (pixels)
  - Single color is assigned to every pixel
  - Color is represented using three primaries – values of basic lights (red, green, blue)
  - Values of primaries stored as unsigned integer numbers

Units

- bit (Binary digIT) - abbreviation „b”- smallest information unit, may represent two-state information YES-NO, 1-0, TRUE-FALSE
- octet (abbrev. ‘o’) – 8 bits, tetrade/nibble – 4 bits
- byte – abbreviation „B” - smallest unit of information addressed by computer – in contemporary g.p computers – 8 bits
- word – unit of information operated on by the computer
  - 1, 2, 4, 8, 16 bytes
- Processor word –unit of information on which a given processor may operate
  - Maximum word size equals register width – in contemporary g.p. computers 64 or 32 bits
- Memory word – unit of information that may be transferred in a single transfer cycle between processor and memory
  - Typ. 128 or 64 bits
  - Longer memory word = faster data transfer

HUMAN CAPITAL
WARSAW UNIVERSITY OF TECHNOLOGY DEVELOPMENT PROGRAMME
EUROPEAN UNION EUROPEAN SOCIAL FUND

Data formats

- Computers operate on words – groups of (usually) 2^n bits
  - Common word lengths in g.p. computers – 8, 16, 32, 64, 128 bits
  - Specialized computers may use different word lengths – 24 bits
- Some computers are also capable of operating on single bits and bit fields
- Data stored as words in memory

Logical (Boolean) data

- Single bit is sufficient to store Boolean data
  - Single bit Boolean format is used in computers providing hardware handling of single bit data
- Boolean data is commonly represented using word of a length natural for a given computer
  - _Bool/bool type in C/C++ is implemented as byte
  - Some other languages use the same size as used for integers – typ. 32 bits
- Representation depends on the programming environment and programming language
  - FALSE is usually represented by a word of value 0
  - TRUE may be represented by
    - Value 1 (C language – result)
    - Any non-zero value (most languages – source arguments)
    - Word made of all bits set to 1 (Visual Basic – result)

HUMAN CAPITAL
WARSAW UNIVERSITY OF TECHNOLOGY DEVELOPMENT PROGRAMME
EUROPEAN UNION EUROPEAN SOCIAL FUND

Unsigned integers

n bits

b_{n-1}	b_{n-2}	b_{n-3}	b_{n-4}	b_{n-5}	b_{n-5}	...	b_5	b_4	b_3	b_2	b_1	b_0

- Natural binary code

$$v_N = \sum_{i=0}^{n-1} b_i \cdot 2^i$$

- BCD (Binary-Coded Decimal)
  - Used for fixed point decimal numbers
  - Decimal digits encoded in binary – 4 bits (nibble/tetrade) per digit
  - Allowed nibble values 0..9
  - Formats:
    - packed – 2 digits per byte; range 0..99
    - Unpacked (“ASCII”) - one digit per byte, range 0..9

Signed integers

- Two's complement

$$v_{2C} = -b_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} b_i \cdot 2^i$$

- Ones' complement

$$v_{1C} = -b_{n-1} \cdot (2^{n-1} - 1) + \sum_{i=0}^{n-2} b_i \cdot 2^i$$

- Sign-magnitude

$$v_{S-M} = (-1)^{b_{n-1}} \cdot \sum_{i=0}^{n-2} b_i \cdot 2^i$$

- Biased
  - typically BIAS = $2^{n-1}-1$

$$v_B = -BIAS + \sum_{i=0}^{n-1} b_i \cdot 2^i$$

Important features of numeric codes

- Representation of zero
  - Number of distinct representations
    - Two representations of 0 in ones complement and sign-magnitude
  - Ease of detection
- Symmetry of range (for signed numbers)
- Representation of sign
  - Ease of sign determination
- Change of sign
  - Ones complement – bit negation
  - Two's complement – bit negation and increment
  - Sign-magnitude – negation of sign bit
- Implementation of arithmetic operations
  - Addition and subtraction of two's complement numbers may be performed using the hardware designed for unsigned numbers
    - Only the overflow detection is different

Numeric codes and their features

Code	*Binary patterns*			*8-bit byte value*		*16-bit word value*		*32-bit word value*	
	min	*zero*	*max*	*min*	*max*	*min*	*max*	*min*	*max*
Unsigned	00...0	00...0	11...1	0	255	0	65535	0	4294967295
2C	10...0	00...0	01...1	-128	127	-32768	32767	-2147483648	2147483647
S-M	11...1	00...0 10...0	01...1	-127	127	-32767	32767	-2147483647	2147483647
1C	10...0	00...0 11...1	01...1	-127	127	-32767	32767	-2147483647	2147483647
Biased	00...0	01...1	11...1	-127	128	-32767	32768	-2147483647	2147483648

HUMAN CAPITAL
HUMAN – BEST INVESTMENT!
WARSAW UNIVERSITY OF TECHNOLOGY
DEVELOPMENT PROGRAMME
EUROPEAN UNION
EUROPEAN
SOCIAL FUND

Change of sign, 2's complement dec→bin conversion

- Change of sign:
  - Ones' complement: ~x
  - Two's complement: ~x+1
  - Biased: BIAS - x
  - Sign-magnitude: negate the sign bit only
  - In codes with asymmetric range (2's C, biased), sign change may cause overflow
- Decimal to 2's complement conversion
  - For non-negative numbers, 2's complement representation is the same as unsigned
    - The representable range limit is smaller – $2^{n-1} – 1$
  - For negative numbers:
    - Convert |x| to unsigned binary
    - Change the sign: negate all bits, then add 1

Fixed point notation

- Obtained by shifting bit weights in integer notation
  - Equivalent to multiplication of integer value by 2^{-f} (f – no. of bits in fractional part)
- Used for 2's complement and unsigned numbers
- Commonly used formats:
  - One or two bits of integer part, rest of word used as fractional part
  - Half of the word used as integer part, the other half as fraction
- Arithmetic operations similar to integer operations
  - Scaling (shifting) required during multiplication and division
  - Do not require special instructions or hardware structures

Fixed point notation - examples

n bits

b_{n-1}	b_{n-2}	b_{n-3}	b_{n-4}	b_{n-5}	b_{n-5}	...	b_5	b_4	b_3	b_2	b_1	b_0

Integer weights, 32-bit word

2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	...	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}

Fixed point weights, 16 bits in integer part, 16 bits in fractional part

2^{15}	2^{14}	2^{13}	...	2^{2}	2^{1}	2^{0}	2^{-1}	2^{-2}	2^{-3}	...	2^{-14}	2^{-15}	2^{-16}

Fixed point weights, 1 bit of integer part, 31 bits in fractional partj

2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	...	2^{-26}	2^{-27}	2^{-28}	2^{-29}	2^{-30}	2^{-31}

Binary vs. decimal fractions - accuracy

- In a fractional part of a binary number, bit weights are 2^{-i}
  - ½, ¼, 1/8, 1/16, 1/32, etc...
- As 10 is not equal to any power of 2, decimal fractions cannot be in general precisely represented as binary fractions with finite number of digits
  - We face the same problem in decimal system:
    1/3 = 0.33333333333333333333333333333333…
  - Only fractions being sums of binary fractions may be represented precisely:
    $0.75 = ¾ = (0.11)_2$
    $0.625 = 5/8 = (0.101)_2$
    ...

Floating point notation – basics

- Examples of decimal FP notation
  $-1.234 \cdot 10^5$ $-0.1234 \cdot 10^6$ $-12.34 \cdot 10^4$
- Elements:
  - Sign of a number
  - Significand (unsigned fixed point number)
  - Exponent (signed integer)
- System base (10) is fixed – there is no need to store it
  - It is only a graphic element of the notation
- Normalized form – form in which integer part of significand is expressed using single non-zero digit
  - In the above example: $-1.234 \cdot 10^5$
  - Other forms are denormalized (non normalized)
  - It is not possible to express zero in a normalized form

HUMAN CAPITAL
WARSAW UNIVERSITY OF TECHNOLOGY DEVELOPMENT PROGRAMME
EUROPEAN UNION EUROPEAN SOCIAL FUND

Binary floating point notation – IEEE754, IEC60559

- If possible, numbers are stored in normalized form
  - Exceptions: zero and numbers of a very small magnitude
  - The only binary digit different from 0 is 1
    - Every normalized number has an integer part of significand equal to 1
    - A non-normalized number has an integer part of significand equal to 0
    - There is no need to store integer part of significand if we know whether the number is normalized or not
- Elements of notation
  - Sign – single bit: 0 – non-negative, 1 – non-positive
  - Exponent field
    - In normalized form – exponent in biased code
    - Special values: 00...00 – denormalized form, 11...11 – Not-a-Number
  - Trailing significand (mantissa) field – contains only fractional part of significand
    - Normalized form – integer part is 1
    - denormalized form – integer part is 0

HUMAN CAPITAL
HUMAN – BEST INVESTMENT!
WARSAW UNIVERSITY OF TECHNOLOGY DEVELOPMENT PROGRAMME
EUROPEAN UNION
EUROPEAN SOCIAL FUND

IEEE 754/IEC60559 – exponent field encoding

- exponent stored in e-bit wide field as biased value
- bias = $2^{e-1} - 1$ (01..11 binary)

exp field	exponent value	meaning
00…00	-bias+1	zero or denormal
00…01	exp-bias = -bias+1	normalized value
0x..xx	exp-bias	normalized value
01..11	exp-bias = 0	normalized value
1x..xx	exp-bias	normalized value
11..10	exp-bias	biggest representable number
11..11	(none)	Not-a-Number

IEEE754/IEC60559 - values

Sign	Biased exponent	Trailing significand (mantissa) field
S	E	T

Sign bit	Exponent field	Exp.value	Mantissa field	Significand value
x	00...00	-bias+1	m	0.m
x	from 00...01 to 11..10	exp-bias	m	1.m
0	11..11	-	00..00	+ infinity
1	11..11	-	00..00	- infinity
x	11..11	-	0x..xx	Signaling NaN
x	11..11		1x..xx	Quiet NaN

IEEE754/IEC60559 – common formats

Word size [b]	Symbolic notation	Field widths		Application
		mantissa	exponent	
16	s10e5	10	5	Computer graphics data storage
24	s16e7	16	7	Computer graphics
16	s7e8	7	8	artificial intelligence (BF16)
32	**s23e8**	**23**	**8**	**binary32 (IEEE single), general purpose**
64	**s52e11**	**52**	**11**	**binary64 (IEEE double), general purpose**
80	s64e15	64	15	Extended double – x87
96	s80e15	80	15	DSP
128	s112e15	112	15	binary128, high-precision arithmetics

HUMAN CAPITAL
WARSAW UNIVERSITY OF TECHNOLOGY DEVELOPMENT PROGRAMME
EUROPEAN UNION EUROPEAN SOCIAL FUND

Floating point arithmetic

- Floating point representation is an approximate one
  - Results of arithmetic operations are also approximate
- Result may depend on order of operations
  - a + b + c may not be the same as c + b + a
  - if |a| << |b|, then a + b may be equal to b
  - Computing the sum or average of many values is not a trivial task
  - Addition and subtraction of multiple arguments should be performed in order of growing magnitude
- Equality test usually gives false result
  - Use abs(a-b) < ε instead
- Precision of IEEE single (24 bits) is smaller than that of 32-bit integer or fixed point
  - Precise representation of integers in IEEE single is limited to absolute value range of 2^{24} only

Memory organization

- In general purpose computers 8-bit byte is the smallest addressable unit of memory
- Multibyte data occupy the appropriate number of consecutive byte-sized memory locations

Multibyte data addressing – byte order

- Little Endian – least significant byte of a data word at the lowest address, more significant bytes follow
- Big Endian – most significant byte at the lowest address

32-bit word

A (MSB)	B	C	D (LSB)

Little-Endian

Addr+0:	D (LSB)
Addr+1:	C
Addr+2:	B
Addr+3:	A (MSB)

Big-Endian

Addr+0:	A (MSB)
Addr+1:	B
Addr+2:	C
Addr+3:	D (LSB)

HUMAN CAPITAL
HUMAN – BEST INVESTMENT!
WARSAW UNIVERSITY OF TECHNOLOGY DEVELOPMENT PROGRAMME
EUROPEAN UNION EUROPEAN SOCIAL FUND

Little-Endian

- Byte numbering corresponds to bit weights
- Natural for computers
- Not so natural for humans
- Type casting preserves pointer value

32-bit word, value = 5

0	0	0	5

32-bit access

Addr+0:	5
Addr+1:	0
Addr+2:	0
Addr+3:	0

16-bit access

Addr+0:	5
Addr+1:	0

8-bit access

Addr+0:	5

Big-Endian

- Natural for humans
- Type cast changes the pointer value
- Fast string compare is possible

“abc” string in memory

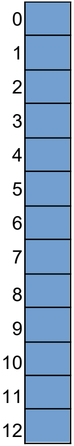
Addr+0:	'a'
Addr+1:	'b'
Addr+2:	'c'
Addr+3:	0

32-bit word read from Addr

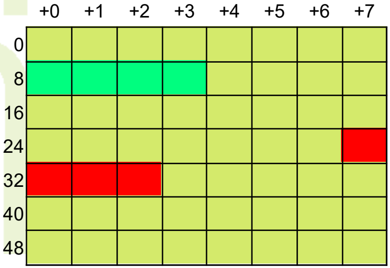
'a'	'b'	'c'	0

Logical vs. physical memory organization

- Logical – vector of bytes



- Physical - “2-dimensional” - bytes grouped in words, usually 32-, 64- or 128-bit
- Bytes within a word may be accessed simultaneously



Data alignment

- Physically the memory is organized as a vector of words, each word being a vector of bytes
  - Any number of bytes within a single word may be accessed at the same time
- Memory word is usually at least as long as the processor's word
  - Typically 64 or 128 bits in 32-/64-bit processors (2 or 4 times longer than processor's word)
  - In older designs (before 1990) memory was sometimes narrower than processor
- The access to multibyte data contained in a single memory word is faster than in case when the data occupies parts of two memory words
  - In the latter case two physical accesses are needed
- New architectures enforce placement of every data item ensuring the fastest possible access

Size alignment

- For the fastest access regardless of the physical memory width, each data item should be placed in memory at the address divisible by its length (rounded up to 2^n)
- Size alignment is enforced by hardware in newer architectures
  - Unaligned access attempt results in execution error
- Even if not enforced in hardware, size alignment boosts the efficiency
  - Compilers enforce size alignment even if underlying hardware doesn't
- Each data type is characterized by two values:
  - Alignment – the data starting address must be a multiple of alignment
    - In C language returned by _Alignof() operator
  - Size – must be a multiple of alignment
    - In C language returned by sizeof() operator
    - in modern environments, size is always 2^n and it is equal to alignment

HUMAN CAPITAL
HUMAN – BEST INVESTMENT!
WARSAW UNIVERSITY OF TECHNOLOGY
DEVELOPMENT PROGRAMME
EUROPEAN UNION
EUROPEAN SOCIAL FUND

Vectors and arrays

- The elements of a vector occupy the consecutive memory locations, starting with the first element (placed at the lowest address – the address of a vector)
  - In C-like languages, the first element has an index of 0
    - v[N] vector elements are v[0], v[1], …, v[N-1]
- Vector alignment is the same as its element alignment
- An n-dimensional array is a vector of (n-1)-dimensional arrays

Vectors and arrays

char s[12];

0	s[0]
1	s[1]
2	s[2]
3	s[3]
4	s[4]
5	s[5]
6	s[6]
7	s[7]
8	s[8]
9	s[9]
10	s[10]
11	s[11]
12	…

char t[3][4];

0	t[0][0]
1	t[0][1]
2	t[0][2]
3	t[0][3]
4	t[1][0]
5	t[1][1]
6	t[1][2]
7	t[1][3]
8	t[2][0]
9	t[2][1]
10	t[2][2]
11	t[2][3]
12	…

int32_t v[12];

0	v[0]
4	v[1]
8	v[2]
12	v[3]
16	v[4]
20	v[5]
24	v[6]
28	v[7]
32	v[8]
36	v[9]
40	v[10]
44	v[11]
48	…

int32_t a[3][4];

0	a[0][0]
4	a[0][1]
8	a[0][2]
12	a[0][3]
16	a[1][0]
20	a[1][1]
24	a[1][2]
28	a[1][3]
32	a[2][0]
36	a[2][1]
40	a[2][2]
44	a[2][3]
48	…

Structures

- Compiler is required to preserve the order of fields
  - It cannot optimize the layout
- Each field must be aligned according to its type requirements
  - Padding (unused space) may be added between fields to achieve the proper alignment
- Structure must be aligned according to the alignment requirement of the field with the biggest alignment
- The structure is padded to the multiple of its alignment
- C language sizeof() operator returns the address distance between two structures of a given type, necessary for allocating a vector of structures.

HUMAN CAPITAL
HUMAN – BEST INVESTMENT!
WARSAW UNIVERSITY OF TECHNOLOGY DEVELOPMENT PROGRAMME
EUROPEAN UNION EUROPEAN SOCIAL FUND

Order of structure fields vs. memory footprint

```
struct st1 {
    char c1;
    double d1;
    short int s1;
    double d2;
    char c2;
    int i1;
    char c3;
    short int s2;
    char c4;
    int i2;
} e1;
```

	+0	+1	+2	+3	+4	+5	+6	+7
0	c1							
8	d1							
16	s1							
24	d2							
32	c2				i1			
40	c3		s2		c4			
48	i2							
56								

sizeof(struct st1) == 56

```
struct st2 {
    double d1, d2;
    int i1, i2;
    short int s1, s2;
    char c1, c2, c3, c4;
} e2;
```

	+0	+1	+2	+3	+4	+5	+6	+7
0	d1							
8	d2							
16	i1				i2			
24	s1		s2		c1	c2	c3	c4
32								

sizeof(struct st2) == 32

Vector data formats

- Contemporary processors may operate on long data words
- Sounds and pictures are expressed using short numbers
- Signal processing algorithms execute the same set of operations on all data samples
- It is possible to treat the long word as a vector of several shorter values
- The introduction of vector data formats enables better utilization of processor's resources

Vector data formats – x86 SSE

- SSE is a vector unit implemented in x86 processors from 1999
- Data word – 128 bits
- Data formats supported (SSE2 and newer versions, 2000→):
  - 16 8-bit integers
  - 8 16-bit integers
  - 4 32-bit integers
  - 2 64-bit integers
  - 4 32-bit IEEE single
  - 2 64-bit IEEE double
  - 8 16-bit s10e5 floats (SSE5)

x86 AVX vector unit

- new vector unit of x86 processors, introduced in 2011 in “2^{nd} generation Intel Core i” processors
- 16 256-bit registers
- FP data formats similar to those of SSE but with 256-bit vectors – 4×64-bit or 8×32-bit
- AVX2 version also supports 256-bit integer vectors – 32×8b, 16×16b, 8×32b, 4×64b
- New extension called AVX512 supports 512-bit vectors

Requirements for the test

- Integer conversion between decimal and 2's complement, sign-magnitude
- Floating point conversion: decimal ↔ IEEE754 binary32
- Structure and vector layout (drawing, sizeof operator value)
- Big-endian/little-endian data placement in memory

ECOAR – Computer Architecture
Lecture notes

End of module

Electrical and Computer Engineering
Faculty of Electronics and Information Technology
Warsaw University of Technology

HUMAN CAPITAL
HUMAN – BEST INVESTMENT!
EUROPEAN UNION
EUROPEAN
SOCIAL FUND

Project is co-financed by European Union within European Social Fund