Lecture 9
Data structures (cont.)

Computing platforms

Novosibirsk State University
University of Hertfordshire

D. Irtegov, A.Shafarenko

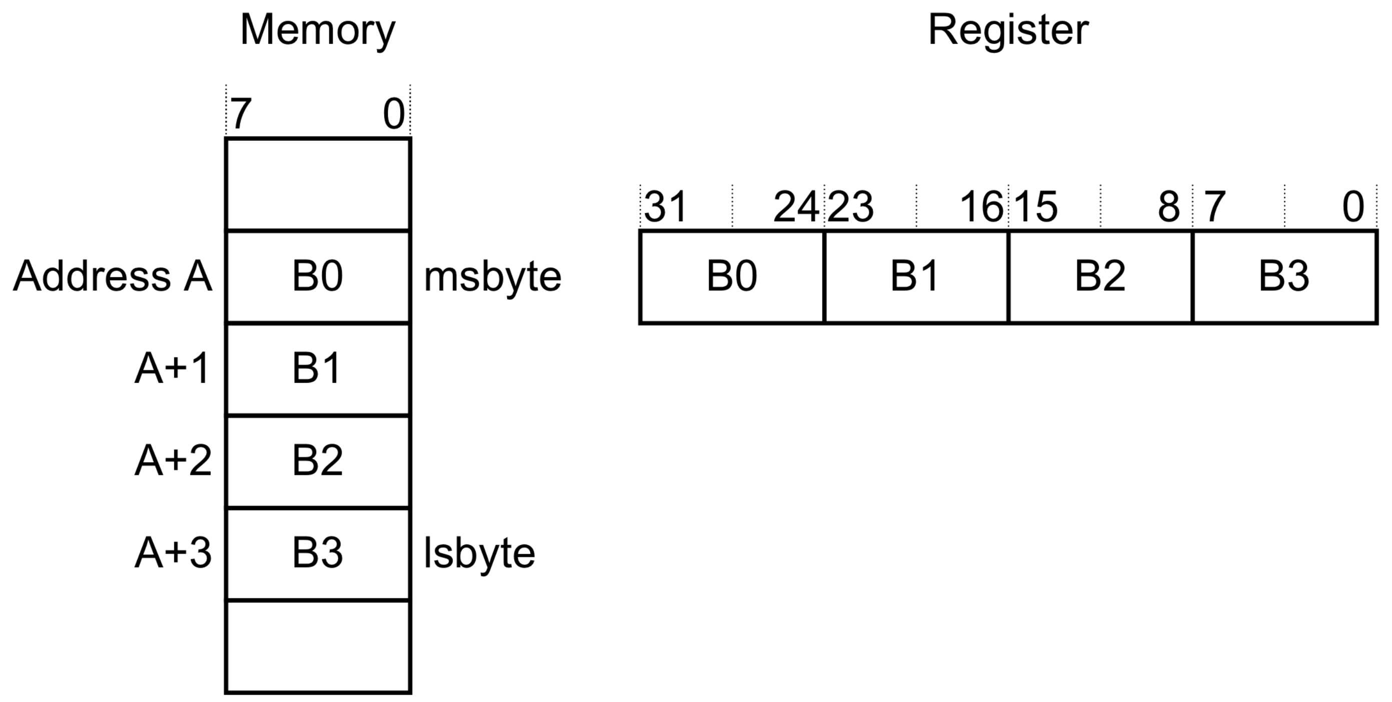
2018

Endianness

- Consider a number containing > 8 bits
- It takes several bytes -> it’s a data structure
- It seems to be simple
- But really, how to layout it in memory?
- There are two logical possibilities
    - (and I know one illogical)

Big endian

- High order byte goes first

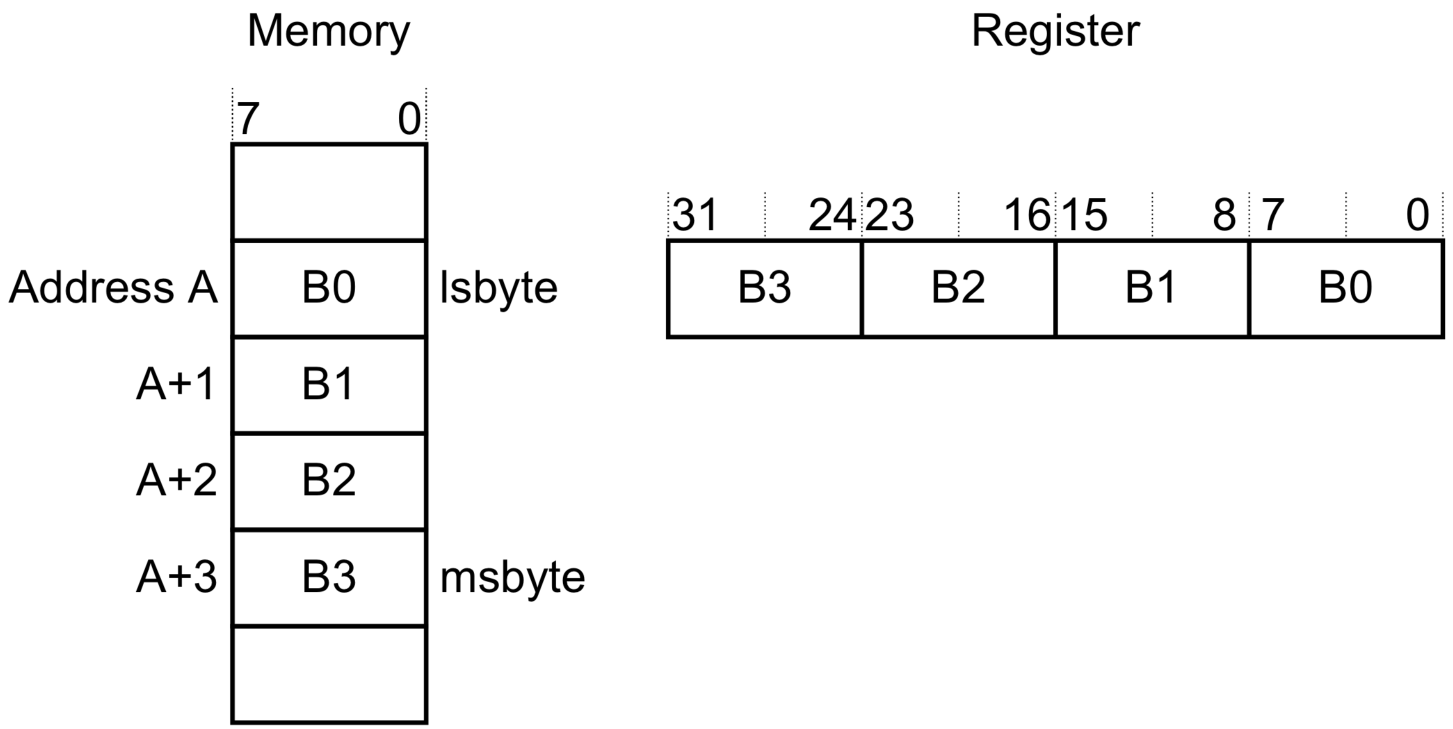


Big endian

- Most famous big-endian ISA: IBM System/360, MC68000
- Advantage:
  - byte-level hex dump reads like number (higher byte on the left)
  - I guess that’s because European got positional numbers from Arabs,
  - So numbers are written in opposite order (relative to text)
- Disadvantage:
  - You cannot convert long * to short *
  - Converted pointer will point to upper half of long
  - This is probably not what you want
  - So many C compilers give a warning when you do such a conversion

Little endian

- Low order byte goes first



Little endian

- Lower byte goes first (at smaller address)
- Most famous little-endian ISA: DEC VAX, x86
- Advantage:
  - Results of long * to short * conversion are intuitively correct
- Disadvantages:
  - It is hard to understand numeric values in byte-level hex dump
  - The code with pointer conversions is not portable to big-endian computers

Mixed endian ISA

- Many popular ISA, including ARM, are mixed endian
- There is a bit in PSW or other configuration register, switching the CPU in big- or little-endian mode
- Typically, a given OS supports only one endianness for all processes
- Otherwise, it would be hard to exchange data between processes and between processes and kernel

Illogical possibility

- PDP-endianness:
  - On PDP-11, 16-bit integers are little-endian
  - But 32-bit integer is big endian (higher half is stored at lower address)
  - PDP-11 was 16-bit, but 32-bit endianness was sort of wired in hardware
  - MUL instruction stored 32-bit result of 16-bit multiplication in PDP-endian
  - PDP-11 32-bit successor, DEC VAX, was fully little endian

History of the term

- Actually a citation from *Gulliver's Travels*
- Oldest document using this term AFAIK is
  IEN 137 (Internet Experiment Note)
  ON HOLY WARS AND A PLEA FOR PEACE
  by Danny Cohen, dated 1 April 1980
  https://www.ietf.org/rfc/ien/ien137.txt
- Indeed, endianness is a big nuisance for
  - network protocols
  - data exchange formats
  - software portability (pointer conversion issue)

Text strings

- Text strings are byte sequences of variable size
- There are two logical possibilities:
  - Using a terminator byte at the end of the string (example: C ASCIIZ strings)
  - Prepending a counter before beginning of the string (example: Pascal strings)

Terminator byte (ASCIIZ)

- In C, 0x00 byte is used as end of string terminator
- In Unix text files, ‘\n’ (0x0A byte) is used as end of line
- In DOS/Windows text files, ’\r\n’ byte pair is used as end of line
  - This dates back to 1960s religious wars on how to interpret ASCII standard
- Advantages:
  - You can have strings of unlimited length
  - For long strings, terminator byte is cheaper (less metadata per volume of text)
- Disadvantage:
  - You need to scan all string to find its length
  - But you often need to know the length, for example to allocate memory for a copy

Counter before the string

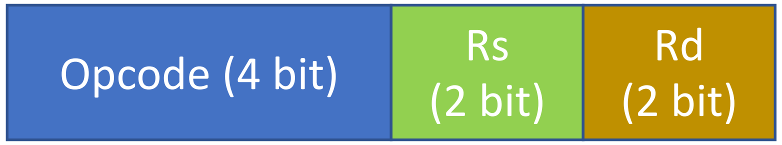
- Disadvantages:
  - Bitness of the counter implicitly limits the size of the string
  - For example, in Pascal, strings are limited by 255 bytes
  - Long (32-bit) counters create significant memory overhead
- Advantages:
  - It is easy to find the size of the string
- Actually, many languages (Python, Java, even C++ STL) use strings with counters

Packed structures (bit fields)

- Consider a situation, when you need to store information which is not multiple of 8 bit
    - Unsigned number between 0 and 63
    - ASCII character (it is actually 7-bit)
    - A set of 300 items
    - CdM-8 register number
- Obvious solution: round the number of bits up to multiple of 8
    - Advantage: data are easy to process
    - Disadvantage: you lose memory

Bit fields

- You often see data formats, when data fields are packed into bytes
- These fields are called bit fields
- Example: CdM-8 instruction with two registers:



- In C, there is support for bit fields in structures
  (but you probably won’t study it)
- In assemblers, there is no support, but you easy can implement them using shifts and bitwise logical instructions

Bit field example in CdM-8 assembly

```
# Get register number from instruction labelled t:
        asect 0
        ldi r1, t
t:      ld r1,r0
        ldi r1,0b00001100
        and r0,r1
        shr r1
        shr r1
        halt
        end
```

Variant records

- Consider a person
- In University, there are two types of people: students and staff
- Students have attributes: name, SRN, course (year of admission), faculty and group
  (in NSU, faculty and year of admission are encoded in group number)
- Staff have attributes: name, payroll number, faculty or department, position
- But sometimes you need a dataset, containing both students and staff

Tagged variant records

- Tagged record has a tag: a field designating the record type
- Tag must be placed on a fixed position in the structure, so any code working with the structure can find the tag
- Variant records can be of fixed or variable length
- C union is *not* a tagged record
  - But you can use struct with tags and union fields to implement a tagged record
- Dynamically typed languages, like Python or JavaScript, usually have type tags attached to all values

Prefix codes

- Prefix code can be seen as sequence of tagged variant records of variable size
- A prefix can be seen as a record tag
  encoding length and structure of following data
- In properly constructed prefix code, entire set of possible encoded values ("codewords") must not contain *any* values that start with any *other* value in the set
- Prefix codes are important because you can uniquely decode them

Prefix codes: examples

- Phone number with area code
  - Local phone number for stationary phones (7 digits in Novosibirsk)
  - Intercity call (Ru): 8-area code-7 digit local number
    (maybe padded with leading 2)
  - No local phone number in Russia can start with 8
  - International call: +Country code-area code-local number
  - Country codes are of variable length and actually use prefix system themselves
  - For example, +7 is formerly USSR numbering scheme.
    In 1990s, most ex-USSR countries took their own country codes.
    Ru and Kz still share 7 prefix, with 76 and 77 codes for Kz,
    70..72 unused and rest of the codes for Ru

Other examples of prefix codes

- Huffman code (you will study it in C programming course)
- Morse code
- UTF-8 encoding
  - Not only prefix, but self-synchronizing
  - If you start decoding from wrong position (not from the prefix), you can detect this and skip to next valid prefix
  - Most other prefix codes do not have this property
- Many machine languages
  - CdM-8 ISA has 1-byte and 2-byte instructions, selected by opcode
  - x86 and VAX ISA have even more complex encoding schemes

UTF-8

Number of bytes	Bits for code point	First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
1	7	U+0000	U+007F	0xxxxxxx			
2	11	U+0080	U+07FF	110xxxxx	10xxxxxx		
3	16	U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
4	21	U+10000	U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

Machine-readable languages

- It is hard to draw a line between complex prefix code and a language
- Machine-readable language must be uniquely decodable
- Natural languages do not have this property
- Prefix codes are uniquely decodable, but most of them are too simple to be considered a language
- Many computer languages actually are not prefix codes
  you need some kind of backtracking to properly parse them
- For example, in C, a + sign is an operator on its own
  and a prefix of ++ and += operators

Parsers

- Writing lexers and parsers is rather complex task,
  so we won’t go into this now
- You will have courses dedicated to this topic
- But it is much simplier than you might think
  - Tools to generate parsers based on formal description of the syntax
  - Regular approaches to manually writing a parser
  - Actually, when you learn rules for describing the syntax,
    you probably can invent such approach yourself
  - Parser libraries for many languages (XML, JSON, yaml)
- You have one task (parsing a text representation of set) which is actually a simple parser