Decimal to non-decimal base

Base Conversions

Convert to base 10 by \textit{multiplication of powers}

\[ 10012_5 = ( \quad )_{10} \]

Convert from base 10 by \textit{repeated division}

\[ 632_{10} = ( \quad )_8 \]

Converting base $x$ to base $y$: convert base $x$ to base 10 then convert base 10 to base $y$
Convert from base 10

More Practice

123_{10} = (   )_3 and check

1234_{10} = (   )_{16} and check

Another way from decimal to base n

from LEFT TO RIGHT, ask “how many” and subtract

(219)_{10} = (   )_2 = (   )_{16}
```c
#include <stdio.h>
int main()
{
    char buf[255] = {0};
    int x = 1234;
    printf("int as hex %x\n", x);
    sprintf(buf,"%x", x);
    printf("buf is %s\n", buf);
    return 0;
}
```
Hex and Binary addition/subtraction

Hex add first, then convert hex to binary and add

- A + 8 =
- 13 + F =
- BEAD + 4321 =

Subtract in hex first, then convert each value to binary and subtract

- 5CD2 – 2A0 =
- 3145 – 1976 =
- A8D2 – 3DAC =

→ carry/borrow 16 each time, since the next place is 16 times as large
(see practice problems 2.4 pg 37)
Addressing and byte ordering

Two conventions (for multi-byte objects)*

What is the address of the object?
What is the order of the bytes in memory?

Typically

Multi-byte objects are stored contiguously
The address of the object is given as the smallest address of the bytes used

- 4-byte integer stored as hex value at address 0x100
- So &x = 0x100, and
- The 4 bytes of x would be stored at memory locations 0x100, 0x101, 0x102, 0x103

* Does not apply to characters because they are single byte values
Addressing and byte ordering (cont)

- **Big Endian**
  - “The big end goes at byte zero”
  - “big end” means the most significant byte of the given value

- **Little Endian**
  - “The little end goes at byte zero”
  - “little end” means the least significant byte of the given value

“Byte zero” means the smallest address used to store the given value

- **Example:** hex/given value is 0x01234567
  - What is the big end of the given value? → 01
  - What is the little end of the given value? → 67
  - What is the lower memory address i.e. byte zero? 0x100

<table>
<thead>
<tr>
<th>Byte order</th>
<th>0x100</th>
<th>0x101</th>
<th>0x102</th>
<th>0x103</th>
</tr>
</thead>
<tbody>
<tr>
<td>Big Endian</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Little Endian</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Big/Little Endian

There is no technological reason to choose one byte ordering convention over the other.

Need to choose a convention and be consistent.

Typically invisible to most application programmers as results are identical.

What if transferring data, though?

Need to know when looking at integer data in memory.

<table>
<thead>
<tr>
<th>Byte order</th>
<th>Big Endian</th>
<th>Little Endian</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x100</td>
<td>01</td>
<td>67</td>
</tr>
<tr>
<td>0x101</td>
<td>23</td>
<td>45</td>
</tr>
<tr>
<td>0x102</td>
<td>45</td>
<td>23</td>
</tr>
<tr>
<td>0x103</td>
<td>67</td>
<td>01</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Byte order</th>
<th>Big Endian</th>
<th>Little Endian</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x103</td>
<td>67</td>
<td>01</td>
</tr>
<tr>
<td>0x102</td>
<td>45</td>
<td>23</td>
</tr>
<tr>
<td>0x101</td>
<td>23</td>
<td>45</td>
</tr>
<tr>
<td>0x100</td>
<td>01</td>
<td>67</td>
</tr>
</tbody>
</table>
Largely, for the same reason you start at the least significant digit (the right end) when you add—because carries propagate toward the more significant digits. Putting the least significant byte first allows the processor to get started on the add after having read only the first byte of an offset.

After you've done enough assembly coding and debugging you may come to the conclusion that it's not little endian that's the strange choice—it's odd that we humans use big endian.

A side note: Humans mostly read numbers and only sometimes use them for calculation. Furthermore we often don't need the exact numbers when dealing with large quantities - taking that into account - big endian is a sensible choice for humans

It reflects the difference between considering memory to always be organized a byte at a time versus considering it to be organized a unit at a time, where the size of the unit can vary (byte, word, dword, etc.)
A look ahead to Chp. 3

Disassembler

A tool that determines the instruction sequence represented by an executable program file

Unix command

- gcc –o hellob hellob.c
- objdump –D –t –s hellob
  -d, --disassemble
    • Display assembler contents of executable sections
  -D, --disassemble-all
    • Display assembler contents of all sections
  -S, --source
    • Intermix source code with disassembly
  -s, --full-contents
    • Display the full contents of all sections requested
  -t, --syms
    • Display the contents of the symbol table(s)
  -T, --dynamic-syms
    • Display the contents of the dynamic symbol table