Characters, Bits and Addresses

CSE 410, Spring 2007
Computer Systems

http://www.cs.washington.edu/410

Readings and References

• Reading
  » Section 2.8, Communicating with People
  » Section 2.9, MIPS addressing

Beyond Numbers

• Most computers today use 8-bit bytes to represent characters
• How many characters can you represent in an 8-bit byte?
  » 256
• How many characters are needed to represent all the languages in the world?
  » a gazillion, approximately (but not a google)

char

• American Standard Code for Information Interchange (ASCII)
  » published in 1968
  » defines 7-bit character codes ...
  » which means only the first 128 characters
  » after that, it’s all “extensions” and “code pages”
• ISO 8859-x
  » codify the extensions to 8 bits (256 characters)
ISO 8859-x

- Each “language” defines the extended chars
  - Latin1 (West European), Latin2 (East European), Latin3 (South European), Latin4 (North European), Cyrillic, Arabic, Greek, Hebrew, Latin5 (Turkish), Latin6 (Nordic)

- Is that enough?
  - For everyone on the planet?

Unicode

- Universal character encoding standard
  - http://www.unicode.org/
- 16 bits should cover just about everything ...
  - “original goal was to use a single 16-bit encoding that provides code points for more than 65,000 characters”
  - the Java char type is a 16-bit character
- How many characters are needed? ...

Unicode UTF-8

Table 3-5 specifies the bit distribution for the UTF-8 encoding form, showing the ranges of Unicode scalar values corresponding to one-, two-, three-, and four-byte sequences. For a discussion of the difference in the formulation of UTF-8 in ISO/IEC 10646, see Section C.3, UCS Transformation Formats.

<table>
<thead>
<tr>
<th>Scalar Value</th>
<th>1st Byte</th>
<th>2nd Byte</th>
<th>3rd Byte</th>
<th>4th Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000-0000FFFF</td>
<td>0xxxxxxx</td>
<td>0yyyyyy</td>
<td>0zzzzzz</td>
<td>01000000</td>
</tr>
</tbody>
</table>

Unicode scalar value:
a number N from 0 to 10FFFF<sub>16</sub> (1,114,111<sub>10</sub>)

Unicode UTF-16

Table 3-4 specifies the bit distribution for the UTF-16 encoding form. Note that for Unicode scalar values equal to or greater than U+10000, UTF-16 uses surrogate pairs. Calculation of the surrogate pair values involves subtraction of 10000<sub>16</sub> to account for the starting offset to the scalar value. ISO/IEC 10646 specifies an equivalent UTF-16 encoding form. For details, see Section C.3, UCS Transformation Formats.

<table>
<thead>
<tr>
<th>Scalar Value</th>
<th>UTF-16</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000000000</td>
<td>0xxxxxxx</td>
</tr>
<tr>
<td>DFFFH</td>
<td>11111111</td>
</tr>
</tbody>
</table>

Where www = uint16 - 1.
Some character URLs

- ANSI X3.4 (ASCII)
  » http://czyborra.com/charsets/iso646.html
- ISO 8859 (International extensions)
  » http://czyborra.com/charsets/iso8859.html
- Unicode
  » http://www.unicode.org/
  » http://www.unicode.org/iuc/iuc10/x-utf8.html

Moving bytes

- A byte can contain an 8-bit character
- A byte can contain really small numbers
  0 to 255\textsubscript{10} or -128\textsubscript{10} to 127\textsubscript{10}
- Sign extension desired effect:
  » sign bit not extended for characters
  » sign bit extended for numbers

Loading bytes

- Unsigned:
  ```
  lbu $reg, a($reg)
  ```
  » the byte is 0-extended into the register
    
    \[
    \begin{array}{cccccccc}
    0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
    \end{array}
    \]
    
    \[\text{xxxx xxxx}\]
- Signed:
  ```
  lb $reg, a($reg)
  ```
  » bit 7 is extended through bit 31
    
    \[
    \begin{array}{cccccccc}
    1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
    \end{array}
    \]
    
    \[1111 1111 \text{xxxx xxxx}\]

Storing bytes

- No sign bit considerations
  » the right most byte in the register is jammed into the byte address given
    
    \[
    \begin{array}{cccccccc}
    0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
    \end{array}
    \]
    
    \[\text{xxxx xxxx}\]
    
    ```
    sb $t0, 2($sp)
    ```
    
    \[
    \begin{array}{cccccccc}
    0 & x & x & x & x & x & x & x \\
    \end{array}
    \]
    
    \[0x7FFEFEFC\]
Storing strings

- Counted strings (for example Pascal strings)
  » byte str[0] holds length: max 255 char
- Counted strings (for example Java strings)
  » int variable holds length: max 2B char
- Terminated strings (for example C strings)
  » no length variable, must count: max n/a
  » Address of string is address of 1st character; first \0 byte signifies end of string

strcpy example

```c
char *strcpy(char *dst, const char *src) {
    char *s = dst;
    while ((*dst++ = *src++) != '\0')
        ;
    return s;
}
```

Software reliability issue: how safe is this?

strcpy compiled

```assembly
strcpy:
    move $v1,$a0
    # remember initial dst
loop:
    lbu $v0,0($a1)       # load a byte
    sb $v0,0($a0)       # store it
    sll $v0,$v0,24       # toss the extra bytes
    addu $a1,$a1,1       # src++
    addu $a0,$a0,1       # dst++
    bne $v0,$zero,loop   # loop if not done
    move $v0,$zero,loop  # loop if not done
    j $v0,$zero,loop  # return initial dst
    move $v0,$zero,loop  # return initial dst
    j $ra # return
```

Manipulating the bits

- Shift Logical
  » sll, srl, slv, srli - shift bits in word, 0-extend
  » use these to isolate bits in a word
  » shift amount in instruction or in register
- Bit by bit
  » and, andi - clear bits in destination
  » or, ori - set bits in destination

Shift to the left, shift to the right, push down, pop up, byte, byte, byte!
Example: bit manipulation

Example of typical application of bit fields

But, note poor choice of field locations
» the received byte is not aligned
» the byte must be shifted before it can be used

To: EE designers of interfaces
» please consider alignment when selecting fields

Multiply and Divide

There is a separate integer multiply unit

Use pseudo-instructions to access

These are relatively slow
» multiply 5-12 clock cycles
» divide 35-80 clock cycles

Addressing modes

Register
Offset + Register
Immediate
PC relative
Pseudodirect
Register only

- Use the 32 bits of the specified register as the desired address
- Can specify anywhere in the program address space, without limitation
- \texttt{jr \$ra}
  » return to caller after procedure completes

Offset + Register

- Specify 16-bit signed offset to add to the base register
- Transfer (\texttt{lw, sw}) base register is specified
  » \texttt{lw} \quad \$t0, 4(\$sp)
  » \texttt{sw} \quad \$t0, 40(\$gp)

Immediate

- The 16-bit field holds the constant value

  \begin{align*}
  0x34080001 & \text{ ori \$8, \$0, 1} & ; 4: \text{ li \$t0,1} \\
  0x3c01ffff & \text{ lui \$1, -1} & ; 5: \text{ li \$t0,-1} \\
  0x3c0100001 & \text{ ori \$8, \$1, -1} & ; 6: \text{ li \$t0,0xFFFF} \\
  0x3c010001 & \text{ lui \$1, 1} & ; 7: \text{ li \$t0,0x1FFF} \\
  0x3c0100000 & \text{ ori \$8, \$1, -1} & ; 8: \text{ li \$t0,0x555AAAA} \\
  0x3c0100000 & \text{ lui \$1, 2854} & ; 9: \text{ li \$t0,main} \\
  0x3c0100000 & \text{ ori \$8, \$1, -1846} & \\
  0x3c010040 & \text{ lui \$1, 64 [main]} & ; 9: \text{ la \$t0,main} \\
  0x3c0100000 & \text{ ori \$8, \$1, 32 [main]} & \\
  \end{align*}

PC relative

- Branch (\texttt{beq, bne}) base register is PC
  » \texttt{beq} \quad \$t0, \$t1, \text{skip}
- The 16-bit value stored in the instruction is considered to be a word offset
  » multiplied by 4 before adding to PC
  » can branch over ± 32 K instruction range
Pseudodirect

- The specified offset is 26 bits long
  - Considered to be a word offset
  - Multiplied by 4 before use
- The top 4 bits of the PC are concatenated with the new 28 bit offset to give a 32-bit address
- Can jump within 256 MB segment