2.9 Communication with People: Byte Data & Constants

- Character
  - Is a byte quantity (00~FF or 0~255)
  - ASCII (American Standard Code for Information Interchange)
    - Page 91, Fig. 2.21
      - 32: space
      - 33: !
      - 34: "
      - 35: # ...
      - 48: 0
      - 49: 1 ...
      - 64: @
      - 65: A
      - 66: B ...
      - 97: a
      - 98: b ...
      - 127: DEL
    - What others?
      - 0~31: control characters
        - 0: ^@ (NUL)
        - 1: ^A
        - 2: ^B ...
        - 7: ^G (BEL)
        - 8: ^H (BS) ...
        - 13: ^M (CR) ...
        - 31: ^- (US, cursor up)
      - 128~255: graphic characters

- String
  - A sequence of characters
  - In “C” language, it is “null-terminated”: “Cab” = 67 97 98 0

How to load/store characters from/to memory? (not “integers”)

Example:
- Copy of the four characters from [160] to [200]
  - Assume $s0=160, $s1=200
  - lw $t0, 0($s0) & sw $t0,0($s1): OK
  - What if it is “Caba” = 67 97 98 97 0?
  - lw $t0, 0($s0) & sw $t0, 0($s1)
  - lw $t1, 4($s0) & sw $t1, 4($s1)
    - we also copied 3 un-wanted bytes in 165-167
    - we want load/store inst. in byte quantity
    - lb & sb
    - the second inst must be
      - lb $t1, 4($s0) & sb $t1, 4($s1)

Byte Data & Constants

- How to load/store characters from/to memory? (not “integers”)
- Example:
  - Copy of the four characters from [160] to [200]
  - Assume $s0=160, $s1=200
  - lw $t0, 0($s0) & sw $t0,0($s1): OK
  - What if it is “Caba” = 67 97 98 97 0?
  - lw $t0, 0($s0) & sw $t0, 0($s1)
  - lw $t1, 4($s0) & sw $t1, 4($s1)
    - we also copied 3 un-wanted bytes in 165-167
    - we want load/store inst. in byte quantity
    - lb & sb
    - the second inst must be
      - lb $t1, 4($s0) & sb $t1, 4($s1)
Byte Data & Constants

- Exercise: `strcpy(a[], b[])`
  - Assume `a[]` starts at 160 ($s0), `b[]` starts at 200 ($s1)
  - We don’t know the string size, but it is null-terminated
  - $zero register can be used

MIPS data transfer instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Comment</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW R3, 500(R4)</td>
<td>Store word</td>
<td>Sign-extended (affect the entire word)</td>
</tr>
<tr>
<td>SH R3, 502(R2)</td>
<td>Store half</td>
<td></td>
</tr>
<tr>
<td>SB R2, 41(R3)</td>
<td>Store byte</td>
<td>Not sign-extended (affect the entire word)</td>
</tr>
<tr>
<td>LW R1, 30(R2)</td>
<td>Load word</td>
<td>(i.e., fill with 0’s in upper 2 or 3 bytes)</td>
</tr>
<tr>
<td>LH R1, 40(R3)</td>
<td>Load halfword</td>
<td></td>
</tr>
<tr>
<td>LHU R1, 40(R3)</td>
<td>Load halfword unsigned</td>
<td></td>
</tr>
<tr>
<td>LB R1, 40(R3)</td>
<td>Load byte</td>
<td></td>
</tr>
<tr>
<td>LBU R1, 40(R3)</td>
<td>Load byte unsigned</td>
<td></td>
</tr>
<tr>
<td>LUI R1, 40</td>
<td>Load Upper Immediate (16 bits shifted left by 16) (later in this lecture)</td>
<td></td>
</tr>
</tbody>
</table>
2.10 MIPS Addressing: Constants & Addressing

- Small constants are used quite frequently (50% of operands)
  e.g., \( A = A + 5; \)
  \( B = B + 1; \)
  \( C = C - 18; \)

- Solutions
  - put 'typical constants' in memory and load them ???
  - create hard-wired registers (like $zero) for constants ???

- MIPS Instructions:
  - `addi $s0, $s0, 4`
  - `slti $s1, $s2, 10`
  - `andi $t1, $t2, 6`
  - `ori $t2, $t2, 4`
  - R-type or I-type?
  - What’s the range of the number?
  - `addi` & `addui`
  - with “`addui`”, the maximum is \(2^{16}-1\)
    \(\Rightarrow\) what if we want a larger constant?

- “i” means immediate data

How about larger constants?

- We'd like to be able to load a 32 bit constant into a register
- Must use two instructions, new "load upper immediate" instruction

  `lui $t0, 0101010101010101`
  `ori $t0, $t0, 1111000011110000`

- Then must get the lower order bits right, i.e.,

  `ori $t0, $t0, 1111000011110000`

- “`addui`” can also be used
- Why “`addi`” can’t be used?
  \(\Rightarrow\) negative !!!
Addresses in Branches and Jumps

• Instructions:
  - `bne $t4,$t5,Label` Next instruction is at Label if $t4!=$t5
  - `beq $t4,$t5,Label` Next instruction is at Label if $t4 =$t5
  - `j Label` Next instruction is at Label

• Formats:

<table>
<thead>
<tr>
<th>I</th>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>16 bit address</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>op</td>
<td></td>
<td></td>
<td>26 bit address</td>
</tr>
</tbody>
</table>

• How to generate a 32-bit address with a 16 or 26-bit number?
  – How do we handle this with load and store instructions?
  => memory address = $rs + 16-bit offset = 32-bit quantity: OK
  – How about in beq/bne?
  – How about in j?

Addresses in Branches

• Instructions:
  - `bne $t4,$t5,Label` Next instruction is at Label if $t4!=$t5
  - `beq $t4,$t5,Label` Next instruction is at Label if $t4 =$t5

• Formats:

<table>
<thead>
<tr>
<th>I</th>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>16 bit address</th>
</tr>
</thead>
</table>

• Next PC = current PC + Label : “PC-relative addressing”
  – PC = program counter
  – most branches are local (principle of locality)
  – Branching range is ???
  • When "current PC"=20000h
    – The jumping address is 20000~2ffff ??? (Sign bit)
    – The jumping address is 20000-7fff ~ 20000+7fff = 18001~27ffh
    => 20000-7fff*4 ~ 20000+7fff*4 = 0 ~ 40000 (-4)
Addresses in Jumps

- **Instructions:**
  
  \[
  j \text{ Label} \quad \text{Next instruction is at Label}
  \]

- **Formats:**

  \[
  j \quad \text{op} \quad \text{26 bit address}
  \]

- **Next PC = First 4-bit of current PC + Label (26-bit)**
  
  - just use high order bits of PC
  
  - address boundaries of 256 MB ???

\[
4 + 26 \neq 30 \neq 32 \quad \text{In effect, 28-bit.} \quad \text{Why ??}
\]

MIPS Addressing Modes

1. Immediate addressing
   
   \[
   \text{addi } \$s0, \$s1, 4
   \]

2. Register addressing
   
   \[
   \text{add } \$s0, \$s1, \$s2
   \]

3. Base addressing
   
   \[
   \text{lw } \$s0, 32(\$s1)
   \]

4. PC-relative addressing
   
   \[
   \text{bne } \$s0, \$s1, \text{Label}
   \]

5. Pseudodirect addressing
   
   \[
   \text{j Label}
   \]
**MIPS Addressing Modes**

- Decoding or reverse engineering (page 153, Fig.3.18)

- Exercise
  - Which instruction is this?
  - 1000 1110 0101 0001 0000 0011 0100
  - $\Rightarrow$ 6-bit + (5-bit + 5-bit + (5-bit + 5-bit + 6-bit))
  - $\Rightarrow$ 100011 : 35 = lw $\Rightarrow$ 6+5+5+16
  - $\Rightarrow$ 100011 10010 10001 00000000 01100100
  - $\Rightarrow$ lw $s2$ $s1$ 100
  - $\Rightarrow$ lw $s1$, 100($s2$)

- But
  - Is memory restricted to 32-bit addressable range or 4GB?
  - What if we have 64-bit memory space?

---

To summarize:

### MIPS operands

<table>
<thead>
<tr>
<th>Name</th>
<th>Example</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 registers</td>
<td>$s0$-$s7$, $t0$-$t9$, $zero$, $s8$-$s15$, $t10$-$t19$, $t20$-$t31$</td>
<td>Fast locations for data. In MIPS, data must be in registers to perform arithmetic. MIPS registers always equal 0. Register $s0$ is reserved for the assembler to handle large constants.</td>
</tr>
<tr>
<td>21 memory words</td>
<td>Memory[0], Memory[1], Memory[2], Memory[3], Memory[4], ...</td>
<td>Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential words differ by 4. Memory holds data structures, such as arrays, and spilled registers, such as those saved on procedure calls.</td>
</tr>
</tbody>
</table>

### MIPS assembly language

<table>
<thead>
<tr>
<th>Category</th>
<th>Instruction</th>
<th>Example</th>
<th>Meaning</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>add</td>
<td>add $s0$, $s1$, $s2$</td>
<td>$s0 = s1 + s2$</td>
<td>Three operands, data in registers</td>
</tr>
<tr>
<td></td>
<td>sub</td>
<td>sub $s0$, $s1$, $s2$</td>
<td>$s0 = s1 - s2$</td>
<td>Three operands, data in registers</td>
</tr>
<tr>
<td></td>
<td>add immediate</td>
<td>add $s0$, $s1$, 100</td>
<td>$s0 = s1 + 100$</td>
<td>Used to add constants</td>
</tr>
<tr>
<td></td>
<td>sub immediate</td>
<td>sub $s0$, $s1$, 100</td>
<td>$s0 = s1 - 100$</td>
<td>Used to subtract constants</td>
</tr>
<tr>
<td>Data transfer</td>
<td>load</td>
<td>lw $s0$, 100($s1$)</td>
<td>Memory $s0 = Memory[0]$</td>
<td>Load from memory to register</td>
</tr>
<tr>
<td></td>
<td>store</td>
<td>sw $s0$, 100($s1$)</td>
<td>Memory[0] = $s0$</td>
<td>Store from register to memory</td>
</tr>
<tr>
<td></td>
<td>load upper immediate</td>
<td>lui $s0$, 100</td>
<td>$s0 = 100 \times 2^{16}$</td>
<td>Load upper immediate</td>
</tr>
<tr>
<td>Conditional branch</td>
<td>branch on equal</td>
<td>beq $s0$, $s1$, 25</td>
<td>$PC = PC + 4 + 100$</td>
<td>Equal test, PC-relative branch</td>
</tr>
<tr>
<td></td>
<td>branch on not equal</td>
<td>bne $s0$, $s1$, 25</td>
<td>$PC = PC + 4 + 100$</td>
<td>Not equal test, PC-relative branch</td>
</tr>
<tr>
<td></td>
<td>set on less than</td>
<td>slt $s0$, $s1$, $s2$</td>
<td>$s0 = 1$; else $s0 = 0$</td>
<td>Compare less than, for beq, bne</td>
</tr>
<tr>
<td></td>
<td>set less than</td>
<td>slti $s0$, $s1$, 100</td>
<td>$s0 = 1$ if $s1 &lt; 100$; else $s0 = 0$</td>
<td>Compare less than constant</td>
</tr>
<tr>
<td>Unconditional jump</td>
<td>jump</td>
<td>j 2500</td>
<td>go to 10000</td>
<td>Jump to target address</td>
</tr>
<tr>
<td></td>
<td>jump register</td>
<td>jr $ra$</td>
<td>go to $ra$</td>
<td>For switch, procedure return</td>
</tr>
<tr>
<td></td>
<td>jump and link</td>
<td>jal 2500</td>
<td>go to 10000</td>
<td>For procedure call</td>
</tr>
</tbody>
</table>