CSE220 – Arrays in MIPS

- Arrays are contiguously allocated bytes of memory to store the data.
- The array name (label) is the starting address of the array in memory (lowest numbered address).
- The total memory used to hold the array is \((\text{# of bytes used to hold the data type}) \times (\text{# of elements in the array})\).
  
  - Eg. char a[10]; each char requires 1 byte of memory, \(1\times 10 = 10\) bytes
  - Eg. int b[50]; each int requires 4 byte of memory (32-bit architecture), \(4\times 50 = 200\) bytes

**Arrays Access**

- When accessing data from an array element in MIPS, we must determine the starting address of the data which you want to access.
  
  - **Ex:** char temp = a[5];
    
    - Assume the address of \(a\) is 0x400.
    - This means \(a[1]\) is at address 0x401, \(a[2]\) at 0x402, … \(a[5]\) at 0x405.
  
  - **Ex:** int temp2 = b[3];
    
    - Assume the address of \(b\) is 0x800.
    - This means \(b[1]\) is at address 0x804, \(b[2]\) at 0x808, and \(b[3]\) at 0x80C.

- There are two ways to access this data via MIPS instructions.
  
  - **Absolute addressing**
    
    - If the index of the array element is “hardcoded”, eg. \(b[3]\), this means the index is always the same value. Therefore in assembly you can directly specify this offset in the instruction itself.
      
      ```
      # $s0 = base address of array b
      la $s0, b    # load the base address of array into register
      lw $t1, 12($s0)  # load the value of \(b[3]\) from memory
      # 12 = # of bytes from base address
      # (size of int * 3)
      ```

  - **Relative address based on a variable**
    
    - If the index of the array element is specified by a variable, eg. \(b[i]\), this means the index is dynamic and therefore must be calculated.
      
      ```
      for (int i = 0; i < 50; i++)
      b[i] = b[i] >> 3;
      ```
# $s0 has base address of b
li $s1, 0        # i = 0
li $t9, 50       # upper bound
loop:
    beq $s1, $t9, done # repeat until i == 50
    sll $t0, $s1, 2    # $t0 = i*4 (multiply # of elements by the size of element)
    add $t0, $t0, $s0 # calc address of element $t0 = addr of array[i]
    lw $t1, 0($t0)     # load value in memory to register, $t1 = array[i]
    sra $t1, $t1, 3    # shift right by 3, $t1 = array[i]/8
    sw $t1, 0($t0)     # store value in memory, array[i] = array[i]/8
    addi $s1, $s1, 1   # increment counter, i = i + 1
    j loop
done:

**Declare Arrays in memory**

- There are multiple ways to declare arrays in memory
  - **.space** assembler directive – declares a number of bytes in memory without initialization
    - Space is just bytes, it can be used to store any type of data. There are no enforced data types in assembly!
    - a: .space 5    # allocates memory for 5 bytes of data
    - b: .space 200  # allocates 50 integers
    - Note: MARS allows you to initialize space using the : operator
      - a: .space 5:’\0’
        # allocates memory for 5 bytes of data and initializes them to NULL
  - **.word** assembler directive – declares integers in memory with initialization
    - c: .word 1,2,3,4,5,6
      # allocates memory for 6 integers (24 bytes) and initializes
      # c[0] = 1, c[1] = 2, . . .
  - **.byte** assembler directive – declares bytes in memory with initialization
    - a: .byte ‘a’,‘b’,‘c’,‘d’,‘e’,‘f’
      # allocates memory for 6 bytes and initializes a[0] = ‘a’,
o .ascii assembler directive – declares bytes in memory initialized to the ASCII characters specified
  ▪ ASCII strings are just arrays of characters!
  ▪ a: .ascii “abcdef”
    # allocates memory for 6 bytes and initializes a[0] = ‘a’,
    # a[1] = ‘b’, . . . .

o .asciiz assembler directive – declares bytes in memory initialized to the ASCII characters specified
  and NULL terminated
  ▪ a: .asciis “abcdef”
    # allocates memory for 7 bytes and initializes a[0] = ‘a’,
    # a[1] = ‘b’, . . . . , a[6] = ‘\0’

• The length of the array or what type of data stored in the array space is not stored or specified anywhere.
  This means it is up to the program to know what type of data it is loading/storing/using.
• This also means that data stored as integers can be accessed by byte or vice versa. And data stored is solely
  0’s and 1’s and it is how you choose to interpret that data that defines its meaning.

ASCII Table
• As we have seen, each character is represented to the computer with a unique bit representation. The
  standard for plaintext is ASCII: American Standard Code for Information Interchange. Java and other more
  modern technologies use the Unicode Standard (allows representation of more characters and other glyphs
  for non-English languages).
• Each character is represented by a unique byte representation:
  o EX: ‘A’ is represented by 01000001₂, which is 0x41 or 65₁₀
  o EX: The NULL character, ‘\0’, is represented by 00000000₂, which is 0x00 or 0₁₀
<table>
<thead>
<tr>
<th>ASCII control characters</th>
<th>ASCII printable characters</th>
<th>Extended ASCII characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 NULL (Null character)</td>
<td>32 space @ 96 ‘</td>
<td>128 Ç 160 á 192 L 224 Ō</td>
</tr>
<tr>
<td>01 SOH (Start of Header)</td>
<td>33 ! 65 A 97 a</td>
<td>129 ŭ 161 í 193 ñ 225 ŕ</td>
</tr>
<tr>
<td>02 STX (Start of Text)</td>
<td>34 “ &quot; 66 B 98 b</td>
<td>130 é 102 ō 194 õ 226 Ō</td>
</tr>
<tr>
<td>03 ETX (End of Text)</td>
<td>35 # 67 C 99 c</td>
<td>131 â 163 ū 195 Ũ 227 Ō</td>
</tr>
<tr>
<td>04 EOT (End of Trans.)</td>
<td>36 $ 68 D 100 d</td>
<td>132 à 164 ň 196 Ō</td>
</tr>
<tr>
<td>05 ENQ (Enquiry)</td>
<td>37 % 69 E 101 e</td>
<td>133 â 165 Ū 197 Ō 229 Ō</td>
</tr>
<tr>
<td>06 ACK (Acknowledgement)</td>
<td>38 &amp; 70 F 102 f</td>
<td>134 à 166 ã 198 à 230 µ</td>
</tr>
<tr>
<td>07 BEL (Del)</td>
<td>39 ‘ 71 G 103 g</td>
<td>135 ç 167 ø 199 Å 231 ñ</td>
</tr>
<tr>
<td>08 BS (Backspace)</td>
<td>40 ( 72 H 104 h</td>
<td>136 ã 168 ç 200 L 232 P</td>
</tr>
<tr>
<td>09 HT (Horizontal Tab)</td>
<td>41 ) 73 I 105 i</td>
<td>137 â 169 © 201 L 233 ŕ</td>
</tr>
<tr>
<td>10 LF (Line feed)</td>
<td>42 * 74 J 106 j</td>
<td>138 ë 170 ñ 202 L 234 Ŷ</td>
</tr>
<tr>
<td>11 VT (Vertical Tab)</td>
<td>43 + 75 K 107 k</td>
<td>139 í 171 ò 203 L 235 Õ</td>
</tr>
<tr>
<td>12 FF (Form feed)</td>
<td>44 , 76 L 108 l</td>
<td>140 i 172 ¼ 204 L 236 Ŷ</td>
</tr>
<tr>
<td>13 CR (Carriage return)</td>
<td>45 - 77 M 109 m</td>
<td>141 í 173 õ 205 M 237 Y</td>
</tr>
<tr>
<td>14 SO (Shift Out)</td>
<td>46 . 78 N 110 n</td>
<td>142 À 174 à 206 ¶ 238 ſ</td>
</tr>
<tr>
<td>15 SI (Shift In)</td>
<td>47 / 79 O 111 o</td>
<td>143 Â 175 ì 207 ± 239 Ž</td>
</tr>
<tr>
<td>16 DLE (Data link escape)</td>
<td>48 0 80 P 112 p</td>
<td>144 E 176 ì 208 ø 240 Ž</td>
</tr>
<tr>
<td>17 DC1 (Device control 1)</td>
<td>49 1 81 Q 113 q</td>
<td>145 æ 177 ñ 209 ð 241 ë</td>
</tr>
<tr>
<td>18 DC2 (Device control 2)</td>
<td>50 2 82 R 114 r</td>
<td>146 ì 178 ë 210 ñ 242 ì</td>
</tr>
<tr>
<td>19 DC3 (Device control 3)</td>
<td>51 3 83 S 115 s</td>
<td>147 ò 179 î 211 ì 243 ¾</td>
</tr>
<tr>
<td>20 DC4 (Device control 4)</td>
<td>52 4 84 T 116 t</td>
<td>148 ô 180 ë 212 ï 244 ¼</td>
</tr>
<tr>
<td>21 NAK (Negative acknowldgement)</td>
<td>53 5 85 U 117 u</td>
<td>149 ó 181 ë 213 ë 245 ï</td>
</tr>
<tr>
<td>22 SYN (Synchronous idle)</td>
<td>54 6 86 V 118 v</td>
<td>150 ì 182 ë 214 ì 246 ¼</td>
</tr>
<tr>
<td>23 ETB (End of trans block)</td>
<td>55 7 87 W 119 w</td>
<td>151 ù 183 ë 215 ì 247 ½</td>
</tr>
<tr>
<td>24 CAN (Cancel)</td>
<td>56 8 88 X 120 x</td>
<td>152 ù 184 ñ 216 é 248 ½</td>
</tr>
<tr>
<td>25 EM (End of medium)</td>
<td>57 9 89 Y 121 y</td>
<td>153 ï 185 õ 217 õ 249 ½</td>
</tr>
<tr>
<td>26 SUB (Substitute)</td>
<td>58 0 90 Z 122 z</td>
<td>154 ì 186 õ 218 õ 250 ½</td>
</tr>
<tr>
<td>27 ESC (Escape)</td>
<td>59 1 91 [ 123 {</td>
<td>155 ã 187 ñ 219 ñ 251 ½</td>
</tr>
<tr>
<td>28 FS (File separator)</td>
<td>60 2 92 \ 124</td>
<td></td>
</tr>
<tr>
<td>29 GS (Group separator)</td>
<td>61 3 93 ] 125 }</td>
<td>157 ã 189 ñ 221 ñ 253 ½</td>
</tr>
<tr>
<td>30 RS (Record separator)</td>
<td>62 4 94 ^ 126 _</td>
<td>158 × 190 ñ 222 ñ 254 ½</td>
</tr>
<tr>
<td>31 US (Unit separator)</td>
<td>63 5 95 _</td>
<td>159 f 191 ñ 223 ñ 255 nbsp</td>
</tr>
</tbody>
</table>