CSE 220:
Systems Fundamentals I
Unit 6:
MIPS Assembly:
Single-dimensional Arrays
Arrays

- Arrays are simply contiguously allocated memory cells
- For each array we need to know:
  - starting address
  - size of an single element in the array (in bytes)
  - number of elements in the array
- The size of the array can be calculated as:
  - # of elements × size of 1 element in bytes
Accessing Arrays

- The first step to accessing an array in memory is to load the **base address** into a register (0x10007000 in example)
- Then, we use an offset to that base address to access individual elements
- In this example we have an array of words
- So each element of the array is 4 bytes away from its neighboring elements

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x10007010</td>
<td>array[4]</td>
</tr>
<tr>
<td>0x1000700C</td>
<td>array[3]</td>
</tr>
<tr>
<td>0x10007008</td>
<td>array[2]</td>
</tr>
<tr>
<td>0x10007004</td>
<td>array[1]</td>
</tr>
<tr>
<td>0x10007000</td>
<td>array[0]</td>
</tr>
</tbody>
</table>
Accessing Arrays

• Java code:  
  ```java
  int[] array = new int[5];
  array[0] = array[0] * 2;
  ```

• MIPS code:  
  ```mips
  # $s0 = array base address
  la    $s0, array
  lw    $t1, 0($s0)    # $t1 = array[0]
  sll   $t1, $t1, 1    # $t1 = $t1 * 2
  sw    $t1, 0($s0)    # array[0] = $t1
  lw    $t1, 4($s0)    # $t1 = array[1]
  sll   $t1, $t1, 1    # $t1 = $t1 * 2
  sw    $t1, 4($s0)    # array[1] = $t1
  ```
Accessing Arrays with a Loop

• Suppose we wanted to implement this C code in MIPS:
  ```c
  int i;
  int array[1000];
  for (i = 0; i < 1000; i++)
      array[i] = array[i] * 8;
  ```

• Assume that $s0$ has been loaded with the base address of the array
Accessing Arrays with a Loop

# $s0 has base address of array
li $s1, 0      # i = 0
li $t2, 1000   # upper bound

loop:
  beq $s1, $t2, done  # repeat until i == 1000
  sll $t0, $s1, 2    # $t0 = i*4
  add $t0, $t0, $s0  # $t0 = addr of array[i]
  lw $t1, 0($t0)     # $t1 = array[i]
  sll $t1, $t1, 3    # $t1 = array[i]*8
  sw $t1, 0($t0)     # array[i] = array[i]*8
  addi $s1, $s1, 1   # i = i + 1
  j loop

done:
Declaring Arrays

• You have two main ways of declaring an array:
  • Give it a name and size in bytes (40 in this example)
    • Example: **ages**: `.space 40`
  • Give it a name and values
    • Example: **ages**: `.word 25, 19, 26, 22, 20`
• In either case it is your job to keep track of the size and/or the number of elements in the array
Character Arrays (Strings)

- Although Java, the Web and other more modern technologies use Unicode to encode characters, MIPS assembly and older languages like C use ASCII: American Standard Code for Information Interchange
- Each text character has unique byte value
  - For example, S = 0x53 (83\textsubscript{10}), a = 0x61 (97\textsubscript{10}), A = 0x41 (65\textsubscript{10})
  - Lower-case and upper-case letters differ by 0x20 (32\textsubscript{10})
- As we saw a little earlier in the semester, the NULL character (0x0) is used to indicate the end of a string
<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</td>
<td>128 Ç</td>
</tr>
<tr>
<td>01 SOH (Start of Header)</td>
<td>64 @</td>
<td>160 á</td>
</tr>
<tr>
<td>02 STX (Start of Text)</td>
<td>34 !</td>
<td>192 L</td>
</tr>
<tr>
<td>03 ETX (End of Text)</td>
<td>66 B</td>
<td>224 Þ</td>
</tr>
<tr>
<td>04 EOT (End of Trans.)</td>
<td>67 C</td>
<td></td>
</tr>
<tr>
<td>05 ENQ (Enquiry)</td>
<td>68 D</td>
<td></td>
</tr>
<tr>
<td>06 ACK (Acknowledgement)</td>
<td>69 E</td>
<td></td>
</tr>
<tr>
<td>07 BEL (Bell)</td>
<td>70 F</td>
<td></td>
</tr>
<tr>
<td>08 BS (Backspace)</td>
<td>71 G</td>
<td></td>
</tr>
<tr>
<td>09 HT (Horizontal Tab)</td>
<td>73 I</td>
<td></td>
</tr>
<tr>
<td>10 LF (Line feed)</td>
<td>74 J</td>
<td></td>
</tr>
<tr>
<td>11 VT (Vertical Tab)</td>
<td>75 K</td>
<td></td>
</tr>
<tr>
<td>12 FF (Form feed)</td>
<td>76 L</td>
<td></td>
</tr>
<tr>
<td>13 CR (Carriage return)</td>
<td>77 M</td>
<td></td>
</tr>
<tr>
<td>14 SO (Shift Out)</td>
<td>78 N</td>
<td></td>
</tr>
<tr>
<td>15 SI (Shift In)</td>
<td>79 O</td>
<td></td>
</tr>
<tr>
<td>16 DLE (Data link escape)</td>
<td>80 P</td>
<td></td>
</tr>
<tr>
<td>17 DC1 (Device control 1)</td>
<td>81 Q</td>
<td></td>
</tr>
<tr>
<td>18 DC2 (Device control 2)</td>
<td>82 R</td>
<td></td>
</tr>
<tr>
<td>19 DC3 (Device control 3)</td>
<td>83 S</td>
<td></td>
</tr>
<tr>
<td>20 DC4 (Device control 4)</td>
<td>84 T</td>
<td></td>
</tr>
<tr>
<td>21 NAK (Negative acknowl.)</td>
<td>85 U</td>
<td></td>
</tr>
<tr>
<td>22 SYN (Synchronous idle)</td>
<td>86 V</td>
<td></td>
</tr>
<tr>
<td>23 ETB (End of trans. block)</td>
<td>87 W</td>
<td></td>
</tr>
<tr>
<td>24 CAN (Cancel)</td>
<td>88 X</td>
<td></td>
</tr>
<tr>
<td>25 EM (End of medium)</td>
<td>89 Y</td>
<td></td>
</tr>
<tr>
<td>26 SUB (Substitute)</td>
<td>90 Z</td>
<td></td>
</tr>
<tr>
<td>27 ESC (Escape)</td>
<td>91 [</td>
<td></td>
</tr>
<tr>
<td>28 FS (File separator)</td>
<td>92 \</td>
<td></td>
</tr>
<tr>
<td>29 GS (Group separator)</td>
<td>93 ]</td>
<td></td>
</tr>
<tr>
<td>30 RS (Record separator)</td>
<td>94 ^</td>
<td></td>
</tr>
<tr>
<td>31 US (Unit separator)</td>
<td>95 _</td>
<td></td>
</tr>
<tr>
<td></td>
<td>127 DEL (Delete)</td>
<td>223 nbsp</td>
</tr>
</tbody>
</table>
Character Arrays (Strings)

- Remember that MIPS uses little-endian
- The example below shows the string **Hello!** stored in memory (0x48 65 6C 6C 6F 21 00)
- The string is *seven* bytes long and extends from address 0x1522FFF0 to 0x1522FFF6
- The null terminator, `\0`, marks the end of the string
- Note how the first character of the string (48) is stored at the lowest byte address

<table>
<thead>
<tr>
<th>Word Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1522FFF4</td>
<td>00 21 6F</td>
</tr>
<tr>
<td>1522FFF0</td>
<td>6C 6C 65 48</td>
</tr>
</tbody>
</table>

Little-Endian Memory
Accessing Characters

• We can use offsets within a word to load or store a byte
• We have three relevant instructions:
  • \texttt{lb}: load byte
  • \texttt{lbu}: load byte unsigned
  • \texttt{sb}: store byte
Accessing Characters

• `lbu $s1, 2($0)` loads register `$s1` with the byte at memory address 2
  • `$s1` now contains `00 00 00 8C`
  • Note that the upper 24 bits are filled with 0s
  • In these examples, usually we will give a register with a non-zero memory address in it (i.e., not `$0$`)
Accessing Characters

- **`lb $s2, 2($0)`** loads register `$s2` with the byte at memory address 2, sign-extending the result
  - `$s2` now contains **FF FF FF 8C**
  - Note that the upper 24 bits are filled with 1s because bit the leftmost bit of 8C is a 1
Accessing Characters

• Suppose that $s3$ contains \texttt{23 45 67 9B}
• \texttt{sb} $s3$, \texttt{3 ($0$)} stores the least significant byte (9B) at memory address 3, replacing F7 with 9B
  • The 3 most significant bytes of $s3$ are ignored
• The word at memory address 0 now contains \texttt{9B 8C 42 03}
Integer Array Examples

• Let’s implement some algorithms for basic array processing:
  • print contents of an array
  • do a sequential search of an array
• Then we’ll write a program that finds and prints the minimum and maximum values stored in an array of integers
Character Array Examples

• Let’s look at some programs involving strings:
  • one that takes a decimal number entered by the user that the program converts to hex
  • another that counts the number of occurrences of a particular target character in a string
  • one containing code that reverses a string
Selection Sort: Java ➔ MIPS

for (int i = 0; i < array.length - 1; i++) {
    // Find the minimum in the array[i..array.length-1]
    int currentMin = array[i];
    int currentMinIndex = i;

    for (int j = i + 1; j < array.length; j++) {
        if (currentMin > array[j]) {
            currentMin = array[j];
            currentMinIndex = j;
        }
    }

    // Swap array[i] with array[currentMinIndex] if nec.
    if (currentMinIndex != i) {
        array[currentMinIndex] = array[i];
        array[i] = currentMin;
    }
}