Example – Loop

Calculate 10!

```
x=1;
j=10;
while (j !=0){
    x = x*j;
    j= j-1;
}
```

```
addi $8, $0, 1
addi $1, $0, 10
L1:mult $8, $1
mflo $8
addi $1, $1, -1
bne $1, $0, L1
```
Less than – slt, slti

\texttt{slt \ $d, \ $s, \ $t}

- compare register 't' and 's'
- $s < $t \Rightarrow \ $d = 1
- $s \geq $t \Rightarrow \ $d = 0
- Notice, no branching instruction, simple comparison
- Need to use this in combination with beq and bne for branching and looping.
# Addressing in Branching

- `beq $5, $6, Exit`

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>16</td>
<td>17</td>
<td>Exit</td>
</tr>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>16 bits</td>
</tr>
</tbody>
</table>

The memory is $2^{32}$ bytes. 16-bits only give us $2^{16}$ addresses – too low!

Thus, the 16-bit value does not specify the exact address, but the relative offset from the current Program Counter.

This implies that we can branch within $+2^{15}$ instructions of the current instruction. ($+2^{17}$ bytes!)

This is good enough for most programs since *if-then-else* statements do not branch beyond a few instructions.
### Addressing in Jumps

- `j  L2`

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10000</td>
</tr>
<tr>
<td>6 bits</td>
<td>26 bits</td>
</tr>
</tbody>
</table>

- J-type instruction format (the 3\textsuperscript{rd} format after R, I)
- Stores the \textbf{exact word} address
  (not relative offset)
- 28-bit byte address
- The upper 4 bits of the address are same as those in PC
- More useful in subroutines – `j`, `jal` etc.
Addressing in Branching and Jumps

• Loop: 
  - `sll $9, $19, 2`
  - `add $9, $t1, $22`
  - `lw $8, 0($9)`
  - `bne $8, $21, Exit`
  - `addi $19, $19, 1`
  - `j Loop`

• Exit:
32-bit constants

• Our emulator provides a special (non-standard MIPS) instruction lis “Load immediate (and) skip”.

• The next word after lis in memory is treated not as an instruction, but a 32 bit piece of data to load into a register, then skip over it (because it's not an instruction).

`lis $d
.word 4254 ;can be +,- decimal or hex`

• Loads the value 4254 into register d. ".word" is an assembler directive to insert a 32 bit value into memory in the middle of our code)

```
0  | lis $d|
4  |  4254 |
8  |  ins 2|
12 |  ins 3|
   |  
```
Assembly File – Suggested Format

- 3 columns
  - label
  - instruction / data
  - comment

- examples

  loop:   lis $1
          .word data ; can use label
  data:   .word 0x20 ; 0x20 = 32
          .word 32
Input / Output

- Special memory addresses are used for keyboard inputs and text outputs.
- To read from user input, read from address 0xFFFF0004, which retrieves characters one at a time (one byte).
- To write to the screen, write to the address 0xFFFF000c. Bytes written to this address get interpreted as ASCII characters, and are printed to the screen.
I/O Example

Read and print back characters until “Esc” key is entered:

```assembly
lis $1
.word 0xFFFF0004   ; Input address
lis $2
.word 0xFFFF000C   ; Output address
addi $3, $0, 0x1B  ; ESC character ascii
.asciiz loop:
lw $4, 0($1)       ; Read char from input
sw $4, 0($2)       ; Print that char to output
bne $3, $4, loop  ; Repeat if ESC not entered
jr $31
```
Iterating through String

Print “CS230”

    lis $1
    .word 0xffffffff000c
    lis $2
    .word print

loop:  lw $3, 0($2)
    beq $3, $0, end
    sw $3, 0($1)
    addi $2, $2, 4
    beq $0, $0, loop

end:   jr $31

print: .word 0x43  ; C
       .word 0x53  ; S
       .word 0x32  ; 2
       .word 0x33  ; 3
       .word 0x30  ; 0
       .word 0x0A  ; LF
       .word 0x00  ; NUL

Print “CS230”