MIPS programming

CS230 Tutorial 05
Multiplication and Division

The multiplication and division instructions use special registers called Hi and Lo. Two instructions allow you to get values from Hi and Lo

- `mfhi $r` --- moves the value in Hi to register $r ("move from Hi")
- `mflo $r` --- moves the value in Lo to register $r ("move from Lo")

**Multiplication**: Hi stores any overflow and Lo stores the actual value.

**Division**: Lo stores the quotient and Hi stores the remainder

What does this mean? After any multiplication, call `mflo` to move the value to a register you can actually access. Same with division, if you want the quotient or remainder.
Intro programming - using MIPS

Using the instructions on the previous slide, we can now write some programs!

Example 1:

Consider the following division: \( \frac{8x^3 + 2x^2 - 11x + 3}{y} \)

Assume that registers $1$ and $2$ have been initialized with values entered using the two ints frontend. Register $1$ contains the value of $x$, register $2$ contains the value of $y$. Write a program that will put the remainder of the division in $3$. Try to minimize the number of registers you use in your solution. You may assume that the value in register $2$ is not $0$. 

Example 1 -- solution

Let’s walk through coding the solution

● Follow along by coding yourself as we solve together
● Think about how to divide the code into small pieces
  ○ For a polynomial this is easy, split it up by the +/- terms
  ○ Test each piece as you go
● Then put the pieces back together
Example 1 -- solution

Solution code:

```assembly
; we know at the outset that $1$ has $x$, and $2$ has $y$
mult $1$, $1$ ; this does $x^2$ and stores it in hi:lo
mflo $3$ ; move the result of the mult into $3$
mult $1$, $3$ ; now we have $x^3$
mflo $4$ ; store $x^3$ in $4$
addi $5$, $0$, $8$ ; store $8$ in $5$
mult $4$, $5$ ; $8x^3$
mflo $4$ ; so now $4$ has $8x^3$
add $3$, $3$, $3$ ; $3 = 3 + 3$, which is $x^2 + x^2$
add $3$, $3$, $4$ ; now we have $8x^3 + 2x^2$ in $3$
addi $4$, $0$, $11$ ; store $11$ in $4$ (since we don't need $4$ anymore)
mult $1$, $4$ ; $11x$
mflo $4$ ; save it
sub $3$, $3$, $4$ ; $3$ is old $3$ ($8x^3 + 2x^2$) - $11x$
addi $3$, $3$, $3$ ; add $3$, so now we have the whole polynomial
div $3$, $2$ ; polynomial / $y$
mfhi $3$ ; store the remainder in $3$
jr $31$
```
Your turn to try some examples

You should try coding these two programs now:

Assume that registers $1$ and $2$ have been initialized with values entered using the two ints frontend. Assuming that variables $m$, $n$, and $p$ refer to the contents of registers $1$, $2$, and $3$ respectively. Write a program to do the following computations:

a. $p = (m + 10) - (n + 20)$

b. $p = (m + n)^2 - 4n$
Solutions ...

Sample solutions for the problems on the previous slide:

a.

\[
\begin{align*}
\text{addi} & \quad 3, \quad 1, \quad 10 \\
\text{addi} & \quad 4, \quad 2, \quad 20 \\
\text{sub} & \quad 3, \quad 3, \quad 4 \\
\text{jr} & \quad 31
\end{align*}
\]

b.

\[
\begin{align*}
\text{add} & \quad 3, \quad 1, \quad 2 \\
\text{mult} & \quad 3, \quad 3 \\
\text{mflo} & \quad 3 \\
\text{addi} & \quad 4, \quad 0, \quad 4 \\
\text{mult} & \quad 2, \quad 4 \\
\text{mflo} & \quad 4 \\
\text{sub} & \quad 3, \quad 3, \quad 4 \\
\text{jr} & \quad 31
\end{align*}
\]
What does this code do?

Writing your own code is one thing, but you also need to be able to read other people’s code and figure out what it’s supposed to do:

Example:

Trace the following programs, and determine their purpose. Assume that registers $1$ and $2$ have been initialized with values entered using the twoints frontend. Assume that variables $m$, $n$, and $p$ refer to the contents of registers $1$, $2$, and $3$ respectively.

```
mult $2, $2
mflo $3
mult $3, $2
mflo $3
add $3, $3, $1
jr $31
```

```
mult $1, $2
mflo $3
addi $4, $0, 7
mult $1, $4
mflo $4
sub $3, $3, $4
addi $3, $3, 5
jr $31
```
Solution to Trace

So what do the programs do?

```
mult $2, $2
mflo $3
mult $3, $2
mflo $3
add $3, $3, $1
jr $31
```

Answer: Computes $p = n^3 + m$

```
mult $1, $2
mflo $3
addi $4, $0, 7
mult $1, $4
mflo $4
sub $3, $3, $4
addi $3, $3, 5
jr $31
```

Answer: Computes $p = mn - 7m + 5$
Labels and Branch instructions

- Labels give a name to a place in the code
  - Tab at the beginning of lines in your code to make labels more readable

```
begin:  addi $3, 0, 1
        sub $3, $3, $1
```

- Branch instructions perform a test (think if statements):
  - If the test succeeds, we go to the given label
  - If it fails, we continue to the next instruction in order
Branch instructions

- There are two branch instructions
  - `beq $s, $t, label` branch to label if $s equals $t
  - `bne $s, $t, label` branch to label if $s not equals $t

- `slt (set less than)`
  - `slt $d, $s, $t` sets $d to 1 if $s < $t, otherwise it sets $d to 0
  - Note: `slt` can cover all of <, ≤, >, ≥ by reordering $s and $t

- There is no `j` instruction!
  - Use `beq $0, $0, label`
Memory instructions

- To read and write from memory we use the `lw` and `sw` instructions.
  - `sw $t, i($s)` stores the value in `$t` in the memory address `$s + i`
  - `lw $t, i($s)` loads the value in the memory address `$s + i` into `$t`

- The `array` frontend is another way to run your mips program, like `twoints`
- Run `array` with `/u/cs230/pub/array program.mips`
- `array` asks for a number (which will go in `$2`), and then that many more numbers. Those numbers will be stored in memory starting at the address that will go in `$1`
Loops: example

Now that we know more about branch and memory instructions, let’s put them together!

Follow along with this example question:

Write a program that sums the elements of the array and places the result in $s_3$. You can assume that the result $x$ satisfies $-(2^{31}) < x < 2^{31}$. 
Array sum in a loop: example

Write a program that sums the elements of the array and places the result in $3. You can assume that the result $x$ satisfies $-(2^{31}) < x < 2^{31}$.

```assembly
; $1 - mem location of start of array
; $2 - num elements in the array
; we want to sum the array and store in $3

   addi $3, $0, 0 ; store 0 in $3 (before starting sum, sum is 0)

; compute the sum in this loop
loop:   beq $2, $0, end ; if there's no elements in the array, done
   lw $4, 0($1) ; load the word in mem location in $1 into $4
          ; this corresponds to the first elt in the array
   add $3, $3, $4 ; add current elt to the sum
   addi $2, $2, -1 ; decrease number of elts left to process
   addi $1, $1, 4 ; now corresponds to address of next elt in array
   beq $0, $0, loop ; this is unconditional jump
          ; equivalent to 'j', but binasm doesn't recognize 'j'

end:    jr $31
```
Assignment reminders

- Always test your code on the university servers before you hand it in
- Hand in the .asm file (*not the .mips file*)