Introduction to MIPS

Lecture 3
Sections 2.1 - 2.4

Robb T. Koether

Hampden-Sydney College

Fri, Aug 30, 2019

Outline

Design Principles

- The four design principles
  - Simplicity favors regularity (and vice versa).
  - Smaller is faster.
  - Make the common case fast.
  - Good design demands good compromises.

Outline

Registers

- There are no variables in assembly code, although there are symbolic constants.
- Why? To keep the hardware simple.
- Operands in assembly code must be in registers or constants (immediate values).
- Registers
  - *Very* fast memory.
  - Limited number of registers (32).
  - “Smaller is faster” design principle.

Registers

- Registers are numbered from 0 to 31.
- Each register holds one word (32 bits).
- They *can* be referred to as `$0`, `$1`, `$2`, ..., `$31`.
- The small number of registers means MIPS code must be carefully written to efficiently use this limited resource.

Registers

Number	Alias	Usage
$0	$zero	Constant 0
$1	$at	Assembler temporary
$2 - $3	$v0 - $v1	Function return values
$4 - $7	$a0 - $a3	Function arguments
$8 - $15	$t0 - $t7	Temporaries
$16 - $23	$s0 - $s7	Saved
$24 - $25	$t8 - $t9	More temporaries
$26 - $27	$k0 - $k1	For OS kernel
$28	$gp	Global pointer
$29	$sp	Stack pointer
$30	$fp	Frame pointer
$31	$ra	Return address

- Each register has an “alias” which indicates its intended use.

Outline

MIPS Instructions

Addition and Subtraction

```
add rd,rs,rt     # rd = rs + rt
sub rd,rs,rt     # rd = rs - rt
```

- Addition and subtraction syntax.
- Four parts
  - Operation name (opcode).
  - Operand receiving the result (destination).
  - First operand for operation (source1).
  - Second operand for operation (source2).

MIPS Instructions

Addition and Subtraction

```
add $t0,$t1,$t2
sub $t3,$t4,$t5

012a4020 = 000000 01001 01010 01000 00000 100000
018d5822 = 000000 01100 01101 01011 00000 100010
```

- The above code will
  - Add `$t2` to `$t1` and store the result in `$t0`.
  - Subtract `$t5` from `$t4` and store the result in `$t3`.
- Note the right-to-left nature of the instructions.

Variables and Registers

- Variables in high-level languages generally have a type.
  - A variable represents a value only of the type it was declared to be.
  - For example, **`int`**s and **`double`**s don't mix.
- Assembly language registers are typeless.
  - Registers store only bits.
  - The operation applied to the register determines how the contents are interpreted.
- However, there is another set of registers, `$f0` through `$f31`, that store floating-point values.
- More about that in a later assignment.

Comments in MIPS

- Use comments to make your code more readable.
- The pound sign (#) is used for MIPS comments.
- Anything from the # to the end of the line is a comment and will be ignored by the assembler.
- Most lines of assembly code should have a comment to describe their purpose.

MIPS Instructions

C Code

```
a = b + c - (d + e);
```

- Compile “by hand” the above C statement into its corresponding MIPS code.
- There is no single MIPS instruction to evaluate the expression.
- Break the statement into several MIPS instructions.

MIPS Instructions

Unstructured C Code

```
temp1 = b + c;
temp2 = d + e;
a = temp1 - temp2;
```

- First, write the C code in *unstructured* C.
- Each line contains exactly one operator, not counting the assignment operator.

MIPS Instructions

MIPS Code

```
add $t0,$s1,$s2          # temp1 = b + c
add $t1,$s3,$s4          # temp2 = d + e
sub $s0,$t0,$t1          # a = temp1 - temp2
```

- The code assumes that
  - `a` is `$s0`,
  - `b` is `$s1`,
  - `c` is `$s2`,
  - `d` is `$s3`,
  - `e` is `$s4`.
- We are not yet accessing memory.

Register zero

- The value zero (`0`) appears frequently in code.
- MIPS dedicates a special register that always contains the value 0.
- Design Principle: “Make the common case fast.”
- The register number is `$0` and its alias is `$zero`.
- Thus, we do not need to load 0 into a register when 0 is needed.
- For branching purposes later, keep in mind that 0 represents “false.”

Register zero

- What can you do with a register always containing the value zero?
- Give the C statement corresponding to

```
add $s0,$s1,$zero
```

- What would you expect this to instruction to do?

```
add $zero,$zero,$zero
```

Register zero

- What can you do with a register always containing the value zero?
- Give the C statement corresponding to

```
add $s0,$s1,$zero
```

- It corresponds to

```
move $s0,$s1
```

- What would you expect this to instruction to do?

```
add $zero,$zero,$zero
```

Register zero

- What can you do with a register always containing the value zero?
- Give the C statement corresponding to

```
add $s0,$s1,$zero
```

- It corresponds to

```
move $s0,$s1
```

- What would you expect this to instruction to do?

```
add $zero,$zero,$zero
```

- It corresponds to

```
nop
```

Immediates

- Immediates are numerical constants in MIPS.
- Immediates often appear in code.
- Following the principle of making the common case fast, MIPS provides special instructions for immediates.
- The immediate value is 16 bits signed, so it must be in the range -32768 to $+32767$.

Incrementing and Decrementing

- There is no increment instruction in MIPS.
- We use the add immediate instruction to increment.

```
addi $s0,$s0,1
```

- Syntax is similar to the `add` instruction.
- This is the “regularity principle” for faster hardware.

Immediates

- Why doesn't MIPS provide a subtract immediate instruction?
- MIPS limits the number of instructions to the absolute minimum.
- Design principle: "Smaller is faster."
- If an operation can be decomposed into simpler instructions, it is not included.

```
subi $s0,$s0,1
```

is the same as

```
addi $s0,$s0,-1
```

Outline

Assignment

Assignment

- Read Sections 2.1 - 2.4.