MIPS Assembly

Review

- A computer has processor, memory, and IO devices.
  - The processor stores values in registers, and modifies values by sending them to the ALU.
  - Memory is where the computer stores a large number of values. Every byte can be accessed by specifying a unique address. The address goes from 0 to 0xffffffff in MIPS. In MIPS we mainly work with word which is 4 bytes.
- MIPS instructions learned:
  - add, sub.
  - lw, sw.

Constant or Immediate Operands

- Many times we use a constant in an operation
  - For example, i++, i--, i += 4, and so on
- Since constant operands occur frequently, we should include constants inside arithmetic operations so that they are much faster
  - MIPS has an add instruction that allows one operand to be a constant
  - The constant must be in 16 bits, as a signed integer in 2’s complement format

```
addi $s1, $s2, 100      # $s1 = $s2 + 100
```

Logical Operations

- Often we need to operate on bit fields within a word.
  - Which allow us to pack and unpack bits into words and perform logical operations such as logical and, logical or, and logical negation

Bit-wise AND

- Apply AND bit by bit
  - The resulting bit is 1 if both of the input bits are 1 and zero otherwise
    - and $t2, $t0, $t1
  - There is also a version of AND with an immediate
    - andi $t2, $t1, 12
    - The immediate is treated as an unsigned 16-bit number
  - Ex:
    - $t0 <- 00001100
    - $t1 <- 00000110
    - $t2 <- 00000100

Bit-wise OR

- Apply OR bit by bit
  - The resulting bit is 1 if at least one of the input bits is 1 and zero otherwise
    - or $t2, $t0, $t1
  - There is also a version of OR with an immediate
    - ori $t2, $t1, 12
    - The immediate is treated as an unsigned 16-bit number
  - Ex:
    - $t0 <- 00001100
    - $t1 <- 00000110
    - $t2 <- 00001110

Bit-wise XOR

- Apply XOR bit by bit
  - The resulting bit is 1 if two bits are different
    - xor $t2, $t0, $t1
  - There is also a version of OR with an immediate
    - xori $t2, $t1, 12
    - The immediate is treated as an unsigned 16-bit number
  - Ex:
    - $t0 <- 00001100
    - $t1 <- 00000110
    - $t2 <- 00001010

NOR

- Since NOT takes one operand and results in one operand, it is not included in MIPS as an instruction
  - Because in MIPS each arithmetic operation takes exactly three operands
  - Instead, NOR is included
    - The resulting bit is 0 if at least one of the input bits is 1
    - nor $t2, $t0, $t1
  - How to implement NOT using NOR?
    - Using $zero as one of the input operands
    - It is included in MIPS as a pseudoinstruction
  - Ex1 (NOT):
    - $t0 <- 00001100
    - $t1 <- 00000000
    - $t2 <- 11110011
  - Ex2 (NOR):
    - $t0 <- 00001100
    - $t1 <- 00000110
    - $t2 <- 11110001

Exercise 1

- How can we load an integer value (like 100) into a register ($t0)?

Exercise 1

- How can we load an integer value (like 100) into a register ($t0)?
  - addi $t0, $zero, 100
  - ori $t0, $zero, 80
  - Which should we prefer?
    - ori. Because it is simpler than add. Simpler means less time, less power consumption.