MIPS Operations/Operands

- "Operation" (instruction) – Produces a value from one or more input values
- "Operand" - Input or Output values for an operation

MIPS operations:
- Arithmetic operations (integer/floating-point) (add, sub, ...)
- Logical operations (and, or, ...)
- Shift operations (shift a certain number of bits to the left or right)
- Compare operations (do something if one operand is less than another, ...)
- Load/stores to transfer data from/to memory
- Branch/jump operations
- System control operations/coprocessor operations

**Question:** Check on the green card. What is the mnemonic for a shift logical left instruction? ___sll___

MIPS operands:
- General-purpose registers: **Question:** name 2 of these ___$t0, $s3___
- Fixed registers, e.g., HI/LO registers
- Memory location
- Immediate value

**Question:** we saw this instruction in lecture: addi $t3,$zero,4. $t3$ is the destination. addi is the mnemonic for the instruction. $zero$ is an operand, and 4 is an operand. What type of operand is $zero$? (pick from the above list): ___Fixed register, since it always contains 0___ What type of operand is 4? ___Immediate___ What do you think “i” stands for in “addi”? ___Immediate___

MIPS Arithmetic

**Arithmetic Type Instruction:**

- `<op>` `<f_destination>` `<f_source1>` `<f_source2>`

**Operands**

All arithmetic instructions have 3 operands:
-Operand order is fixed: destination first
- 32 registers (page 2 of green card)

**Question:** What number is $t3$? ___11___ What number is $s3$? ___19___

**Examples**
- add $t0, $s0, $s2 # $t0 = $s0 + $s2
- sub $s0, $t0, $t1 # $s0 = $t0 – $t1

**Question:** In the sub instruction, which register is rd? ___$s0___ Which register is rs? ___$t0___ Which register is rt? ___$t1___
General-Purpose Registers

• GPR: all can be used as operands in instructions

• Still, conventions and limitations exist to keep GPRs from being used arbitrarily
  – r0, termed $zero, always has a value “0”
  – r31, termed $ra (return address), is reserved for subroutine call/return
  – Etc. (we’ll see other conventions/limitations later)
  – Register usage and related software conventions are summarized in “application binary interface” (ABI), which is important when writing system software such as an assembler and a compiler

• Question: Check the green card: what is the calling convention for $t0-$t7? Temporaries Note that these conventions are part of the ABI mentioned above. What does ABI stand for? Application Binary Interface

Question: R-Format Example

• add $8,$9,$10

Translate the above instruction. Specifically: Look up “add” on green card. As you can see, “add” is R format. Look up the R format and the opcode/funct values for add. Then, fill in the tables and the underline below.

Decimal number per field representation:

<table>
<thead>
<tr>
<th>0</th>
<th>9</th>
<th>10</th>
<th>8</th>
<th>0</th>
<th>32</th>
</tr>
</thead>
</table>

Binary number per field representation (use the right # of bits!):

<table>
<thead>
<tr>
<th>000000</th>
<th>01001</th>
<th>01010</th>
<th>01000</th>
<th>00000</th>
<th>10000</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

hex representation: __0x012A4020
Question: I-Format Example

- MIPS Instruction:
  addi  $8,$9,7  $8 is rt; $9 is rs. This instruction adds 7 to the contents of $9 and stores it in $8.

Translate the instruction above to fill in the following tables:

**Decimal number per field representation:**

| 8 | 9 | 8 | 7 |

**Binary number per field representation:**

<table>
<thead>
<tr>
<th>001000</th>
<th>01001</th>
<th>01000</th>
<th>0000000000000111</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>8    0   0   0   7</td>
</tr>
</tbody>
</table>

**Hex representation:** 0x21280007

Answer: Verilog

(2) SignExtImm = {16(immediate[15]), immediate}

{x,y,z} means xyz (the three concatenated together)
{3{a}} means a repeated 3 times: aaa
Immediate[15] means bit 15 of the immediate field of the instruction

{16(immediate[15]), immediate}

(1) The top bit of the immediate field
(2) Repeated 16 times
(3) Followed by the immediate field

Our specific example: 00000000000000000000000111
Answer: Exercise

Which instruction has same representation as $35_{\text{ten}}$?

<table>
<thead>
<tr>
<th>Option</th>
<th>Instruction</th>
<th>Binary</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>add $0, 0, 0$</td>
<td>0 0 0 0 0 32</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>subu $s0,$s0,$s0</td>
<td>0 16 16 16 0 35</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>lw $0, 0($0)</td>
<td>35 0 0</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>addi $0, 0, 35</td>
<td>8 0 0 35</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>subu $0, 0, 0$</td>
<td>0 0 0 0 0 35</td>
<td></td>
</tr>
</tbody>
</table>

F. Trick question! Instructions are not numbers. (yes they are!!!)

Answer is E.

Note: Registers numbers and names:
0: $s0$, 8: $t0$, 9:$t1$, ..., 16: $s0$, 17: $s1$, ...

Note: Opcodes and function fields
- add: opcode = 0, function field = 32
- subu: opcode = 0, function field = 35
- addi: opcode = 8
- lw: opcode = 35

---

Answer: Exercise

- What values are placed into $t0$, $t1$, $t5$ and $t4$ by the following pieces of code?

addi $t0,$0,0xA23 $t0=00000000000000000000101000100011$

addi $t1,$0,0x80001B42 $t1=10000000000000000001101101000010$

and $t4,$t0,$t1 $t4=0000000000000000000000001010000010$ 0x00000A02

or $t5,$t0,$t1 $t5=1000000000000000000000001101101100011$ 0x80001B63
Sample Program

.data # sample0.asm
.word
.c:    3
.k:    5
.text
    la $t0,c       # address of c
    la $t1,k       # address of k
    lw $s0,0($t0) # load the contents of c
    lw $s1,0($t1) # load the contents of k
    slt $s3,$s0,$s1 # if $s0 < $s1 then $s3 = 1; else $s3 = 0
    beq $s3,$0,notless #if $s3 == 0: go to notless; o/w just go to the next instruction
    sw $s0,0($t1)  #store contents of c into k
    sw $s1,0($t0)  #store the contents of k into c
notless:
    # the end of the code; we just stop here

# QUESTION: So, what did we do in this program?
If c < k then we swapped their values. If not, we just left them alone.