MIPS Instructions Cheat Sheet

Ahmed Ayman

Registers

> MIPS has a 32 × 32-bit register file -> 32 Register Each one has 32 BITS

```
# Name              # OP-CODE   # Description
# ____________________________________________________________________ #
$zero               # 0         # stores the value 0 - can't be changed
$at                 # 1         # reserved for the assembler
$v0 - $v1           # 2 - 3     # Proc return values and exp eval
$a0 - $a3           # 4 - 7     # Proc arguments
$t0 - $t7           # 8 - 15    # for temporary values
$s0 - $s7           # 16 - 23   # general purpose saved variables
$t8 - $t9           # 24 - 25   # for temporary values
$k0 - $k1           # 26 - 27   # reserved for the OS
$gp                 # 28        # Global Pointer
$sp                 # 29        # Stack Pointer
$fp                 # 30        # Frame Pointer
$ra                 # 31        # Proc Return Address
```

Arithmetic Operations Instructions

- Three Operands, two srcs one dest
- No Immediate Operands

```
add dest, src1, src2        # dest = src1 + src2
sub dest, src1, src2        # dest = src1 - src2
```

- Only one Immediate Operand - src2

```
addi dest, src1, c
    # No subtract Immediate Instruction, use negative constant with addi
addi dest, src1, -c
    # move data between Registers
    # NO MOV INSTRUCTION use addition with zero
add dest, $t1, $zero
addi dest, $t1, 0
```

Logical Operations Instructions

```
and dest, src1, src2        # dest = src1 & src2 - performs bitwise AND between src1 and src2 and stores the result in dest
or dest, src1, src2         # dest = src1 | src2 - performs bitwise OR between src1 and src2 and stores the result in dest
nor dest, src1, src2        # dest = ~(src1 | src2) - performs bitwise NOR between src1 and src2 and stores the result in dest
# NO NOT INSRUCTION , USE NOR
nor dest, dest, dest        # dest = ~(dest | dest) = ~dest - negeate dest and stores it in dest

andi dest, src1, c      # and immediate - dest = src1 & c - performs bitwise AND between src1 and c and stores the result in dest
ori dest, src1, c       # or immediate - dest = src1 | c - performs bitwise OR between src1 and c and stores the result in dest

sll dest, src1, c       # shift logical left - dest = src1 << c - performs shift left by c on src1 and stores the result in dest - same as multiplying by 2^c
sll dest, src1, c       # shift logical left - dest = src1 >> c - performs shift right by c on src1 and stores the result in dest - same as dividing by 2^c
```

Data Transfer Instructions

- Two Operands, one srcs one dest
- $s1 + c must be divisible by 4

```
lw dest, c (base)       # Load word - Loads a word from Mem[base + c] into dest - base is the base address, c is the offset, dest is dest
sw dest, c (base)       # Store word - Stores a word from dest into Mem[base + c] - base is the base address, c is the offset, dest is dest

lh dest, c (base)       # Load half - Loads halfword from Mem[base + c] into dest - base is the base address, c is the offset, dest is dest
lhu dest, c (base)      # Load unsigned half- Loads unsigned halfword from Mem[base + c] into dest  - base is the base address, c is the offset, dest is dest
sh dest, c (base)       # Store half - Stores halfword from dest into Mem[base + c]  - base is the base address, c is the offset, dest is dest

lb dest, c (base)       # Load byte - Loads byte from Mem[base + c] into dest  - base is the base address, c is the offset, dest is dest
lbu dest, c (base)      # Load unsigned byte - Loads unsigned byte from Mem[base + c] into dest  - base is the base address, c is the offset, dest is dest
sb dest, c (base)       # Store byte - Stores byte from dest into Mem[base + c] - base is the base address, c is the offset, dest is dest

ll dest, c (base)       # Load linked word - Loads word as 1st half of atomic swap from Mem[base + c] into dest  - base is the base address, c is the offset, dest is dest  - dest = Mem[base + c]
sc dest, c (base)       # Store condition word - Stores word as 2nd half of atomic swap from dest into Mem[base + c]  - base is the base address, c is the offset, dest is dest - Mem[base + c] = dest

lhu dest, c             # Load upper immediate - Loads c into upper 16 bits of dest - dest = c * 2^16
```

Jumps

Conditional Jumps

- If condition is true : jump to label. else : continue sequentially

```
beq src1, src2, L       # branch if equal - Jump to L if src1 == src2
bne src1, src2, L       # branch if not equal - Jump to L if src1 != src2
```

Unconditional Jumps

- Jump to label/register anyway

```
j L                     # Unconditional Jump - jump to L
jal L                   # Jump and Link - Jump to proc label L and store the next line address in $ra - used to call a proc
jr $ra                  # Jump Register - Jump to the address in $ra by copying $ra value to the program counter - used to return from proc
```

Conditionals

```
slt dest, src1, src2    # Set Less Than - set dest if src1 < src2 else clear dest
slti dest, src1, c      # Set Less Than Immediate - set dest if src1 < c else clear dest

sltu dest, src1, src2   # Set Less Than unsigned  - set dest if src1 < src2 else clear dest
sltui dest, src1, c     # Set Less Than unsigned Immediate - set dest if src1 < c else clear dest
```