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 $t0, $s0, $s1       # $t0 = $s0 + $s1
sub $t0, $s0, $s1       # $t0 = $s0 - $s1
```

- Only one Immediate Operand - src2

```
addi $t0, $s0, 4
    # No subtract Immediate Instruction, use negative constant with addi
addi $t0, $s0, -1
    # move data between Registers
    # NO MOV INSTRUCTION use addition with zero
add $t0, $t1, $zero
addi $t0, $t1, 0
```

Logical Operations Instructions

```
and $t0, $s0, $s1       # $t0 = $s0 & $s1 - performs bitwise AND between $s0 and $s1 and stores the result in $t0
or $t0, $s0, $s1        # $t0 = $s0 | $s1 - performs bitwise OR between $s0 and $s1 and stores the result in $t0
nor $t0, $s0, $s1       # $t0 = ~($s0 | $s1) - performs bitwise NOR between $s0 and $s1 and stores the result in $t0
# NO NOT INSRUCTION , USE NOR
nor $t0, $t0, $t0       # $t0 = ~($t0 | $t0) = ~$t0 - negeate $t0 and stores it in $t0

andi $t0, $s0, c        # and immidiate - $t0 = $s0 & c - performs bitwise AND between $s0 and c and stores the result in $t0
ori $t0, $s0, c         # or immidiate - $t0 = $s0 | c - performs bitwise OR between $s0 and c and stores the result in $t0

sll $t0, $s0, c         # shift logical left - $t0 = $s0 << c - performs shift left by c on $s0 and stores the result in $t0 - same as multiplying by 2^c
sll $t0, $s0, c         # shift logical left - $t0 = $s0 >> c - performs shift right by c on $s0 and stores the result in $t0 - same as dividing by 2^c
```

Data Transfer Instructions

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

```
lw $s0, c ($s1)         # Load word - Loads a word from Mem[$s1 + c] into $s0  - $s1 is the base address, c is the offset, $s0 is dest
sw $s0, c ($s1)         # Store word - Stores a word from $s0 into Mem[$s1 + c] - $s1 is the base address, c is the offset, $s0 is dest

lh $s0, c ($s1)         # Load half - Loads halfword from Mem[$s1 + c] into $s0 - $s1 is the base address, c is the offset, $s0 is dest
lhu $s0, c ($s1)        # Load unsigned half- Loads unsigned halfword from Mem[$s1 + c] into $s0  - $s1 is the base address, c is the offset, $s0 is dest
sh $s0, c ($s1)         # Store half - Stores halfword from $s0 into Mem[$s1 + c] - $s1 is the base address, c is the offset, $s0 is dest

lb $s0, c ($s1)         # Load byte - Loads byte from Mem[$s1 + c] into $s0  - $s1 is the base address, c is the offset, $s0 is dest
lbu $s0, c ($s1)        # Load unsigned byte - Loads unsigned byte from Mem[$s1 + c] into $s0  - $s1 is the base address, c is the offset, $s0 is dest
sb $s0, c ($s1)         # Store byte - Stores byte from $s0 into Mem[$s1 + c]  - $s1 is the base address, c is the offset, $s0 is dest

ll $s0, c ($s1)         # Load linked word - Loads word as 1st half of atomic swap from Mem[$s1 + c] into $s0  - $s1 is the base address, c is the offset, $s0 is dest  - $s0 = Mem[$s1 + c]
sc $s0, c ($s1)         # Store condition word - Stores word as 2nd half of atomic swap from $s0 into Mem[$s1 + c]  - $s1 is the base address, c is the offset, $s0 is dest - Mem[$s1 + c] = $s0

lhu $s0, c              # Load upper immediate - Loads c into upper 16 bits of $s0 - $s0 = c * 2^16
```