armlet quick reference

Our first `armlet` program

```
        mov $0, 10
        mov $1, 1
        mov $2, 0
@loop:
        cmp $0, 0
        beq >done
        add $2, $2, $1
        add $1, $1, 1
        sub $0, $0, 1
        jmp >loop
@done:
        hlt
```

```
var t = 10
var i = 1
var s = 0
while(t != 0) {
   s = s + i
   i = i + 1
   t = t - 1
}
```

Instruction set

(Instructions controlling the data path)

```
nop                     # no operation
mov $L, $A              # $L = $A (copy the value of $A to $L)
and $L, $A, $B          # $L = bitwise AND of $A and $B
ior $L, $A, $B          # $L = bitwise (inclusive) OR of $A and $B
eor $L, $A, $B          # $L = bitwise exclusive-OR of $A and $B
not $L, $A              # $L = bitwise NOT of $A
add $L, $A, $B          # $L = $A + $B
sub $L, $A, $B          # $L = $A - $B
neg $L, $A              # $L = -$A
lsl $L, $A, $B          # $L = $A shifted to the left by $B bits
lsr $L, $A, $B          # $L = $A shifted to the right by $B bits
asr $L, $A, $B          # $L = $A (arithmetically) shifted to the right by $B bits
```

```
mov $L, I               # $L = I (copy the immediate data I to $L)
add $L, $A, I           # $L = $A + I
sub $L, $A, I           # $L = $A - I
and $L, $A, I           # $L = bitwise AND of $A and I
ior $L, $A, I           # $L = bitwise (inclusive) OR of $A and I
eor $L, $A, I           # $L = bitwise exclusive OR of $A and I
lsl $L, $A, I           # $L = $A shifted to the left by I bits
lsr $L, $A, I           # $L = $A shifted to the right by I bits
asr $L, $A, I           # $L = $A (arithmetically) shifted to the right by I bits
```

```
loa $L, $A              # $L = [contents of memory word at address $A]
sto $L, $A              # [contents of memory word at address $L] = $A
```

Instruction set

(Instruction controlling the flow of execution)

`cmp $A, $B` `# compare $A (left) and $B (right)` `cmp $A, I` `# compare $A (left) and I (right)`	Comparison

`jmp $A` `# jump to address $A`
`beq $A` `# ... if left == right (in the most recent comparison)`
`bne $A` `# ... if left != right`
`bgt $A` `# ... if left > right (signed)`
`blt $A` `# ... if left < right (signed)`
`bge $A` `# ... if left >= right (signed)`
`ble $A` `# ... if left <= right (signed)`
`bab $A` `# ... if left > right (unsigned)`
`bbw $A` `# ... if left < right (unsigned)`
`bae $A` `# ... if left >= right (unsigned)`
`bbe $A` `# ... if left <= right (unsigned)`
	Jump and branch based on the results of the latest comparison

`jmp I` `# jump to address I`
`beq I` `# ... if left == right (in the most recent comparison)`
`bne I` `# ... if left != right`
`bgt I` `# ... if left > right (signed)`
`blt I` `# ... if left < right (signed)`
`bge I` `# ... if left >= right (signed)`
`ble I` `# ... if left <= right (signed)`
`bab I` `# ... if left > right (unsigned)`
`bbw I` `# ... if left < right (unsigned)`
`bae I` `# ... if left >= right (unsigned)`
`bbe I` `# ... if left <= right (unsigned)`
	Jump and branch based on the results of the latest comparison

`hlt` `# halt execution` `trp` `# trap (break out of execution for debugging)`	Halt