MIPS

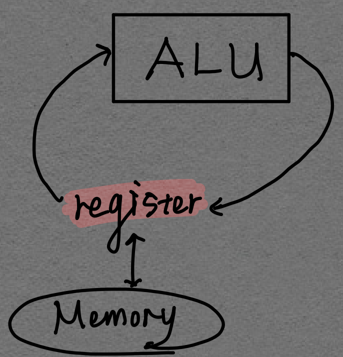
limited to 2^{32} bytes

nano second
30cm
processor -> tasks.

CPU - simple operation
- once per time per CPU/core

Memory - a 1D array of bytes
lastword 0xffff_fffc

32 registers, 32 bytes each



Basic instructions

- add / sub / and / or
- nor - true if both false (not or)
  xor - exclusive or   not equal
- not - nor itself
  * xor with 1111...1111

```
lw  $s0  0($s1)
```
update / offset / register

- word 4 bytes
- lh load Half-word
- lb load Bite

```
sw  $s0, 0($s1)      # save s0 to s1
```
~~write here~~ ~~read $s1, add 0 -> address~~

```
my_var :  .word  123
```
label — storage directive — allocate 32 bites in memory (.byte .half)

```
int my-var = 123;
```

```
* my-var:
    another:
      .word  123
```
} store 123 address to both

*

```
la  $s0, my-var        # s0 = &my-var
    WORD *s0 = &my-var;
lw  $s1, 0($s0)        # $1 = my-var
```

```
.data
count :   .half  0
.text
main :
        # code here
```

* can be used multiple times.

jump & link

```
bne  $s0 , $s1 , ELSE          addi $t0 , $zero , 1   # t0 = 1

jal  doThis
J    doThat
jump
ELSE :
jal  doThat

DONE :
jal  doNext
```

syscall

```
addi  $v0, $zero, 1       # print_int
      what operation  print int
addi  $a0, $zero, 123     # print_int(123)
      what parameter
syscall
```
} printf("%d", 123);

0xa '\n'

$v0	Use	$a0
1	print int	int
4	string	char*
11	char	char
10	exit()	n/a

char* — array, string of char — char pointer of array[0]

* Textbook A-44

printf("%s", "foobar");

printf("%c", "x"); — 'x' -> 120, 0x78

```
addi  $v0, $zero, 11
addi  $a0, $zero, 0x78    'x'
syscall
```

where we are, what we know

LOOP

okay to use indentation

addi $t0, $zero, 0 (clearer)

use slti not beq (dangerous)