1.2) 1280
×800

1.2.1) → × 8.3 = 24.576.000 | 8
3.072.000 //

1.2.2) 2.1024.1024.1024 | 3.072.000 ← 1280 × 800
anterior) M K B 699 // 699 frames

a) 2.1024.1024.1024 | 921.600 ← 640 × 480
3330 frames //

b) 2.1024.1024.1024 | (1024.768.3) ← 1024 × 768
$2.2^{10}.2^{10}.2^{10} = 2^{31}$
210 frames //

1.2.3) $2^8.2^{10}.2^3 = 2^{21}$ | $100.2^{10}.2^{10}$

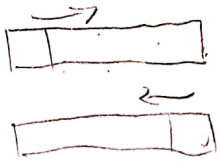
a) $\frac{2^{21}}{100.2^{20}} = \frac{2}{100} = 0,02$ segundos //

b) $\frac{2^{21}}{2^{30}} = \frac{1}{2^9} = 0,00195$ segundos //

1.2.4) ms
5 ——— 1 50 ——— 1 b) 20 μs //
2000 ——— 400 x ——— 400
7 ——— 1 a) 20 μs //
2000 ——— x → 2000/7

1.2.5) 5.10^5 ——— 1 b) 5,714285,714
x ——— 400 a) → 2 segundos // 5,714 segundos //

1.2.6) 5000 ——— 1 b) 4,285 ms //
x ——— 400 → 2 ms //
a)



2.4.1)
LW $t0, 16($s7)
ADD $s0, $s1, $s2
ADD $s0, $s0, $t0

2.4.2) 3 instruções

2.4.3) 5 registradores

2.5.2) * no final.
b) for(int i=0; i<5-1; i++){
  for(int j=i+1; j<5; j++){
    if(A[i]<A[j]){
      swap(A[i], A[j]);

LOOP:
ADDi $t0, $gp0, $gp0
ADDi $t1, $gp0, $gp0

2.5.5)
a) little-endian

endereço	0	4	8	12	16
dado	5	4	3	2	1

big-endian

endereço	0	4	8	12	16
dado	1	2	3	4	5

b) little-endian

endereço	0	4	8	12	16
dado	6	4	3	2	1

big-endian

endereço	0	4	8	12	16
dado	1	2	3	4	6

2.6.5) +5

a) $s0 = 10
$s1 = 20
$s2 = 1
$s3 = 40

$s0 = 10 - 20 = -10
$s0 = (-10) - 40 = -50
$s0 = (-50) + 20 = -30
$s0 = 0xFFFFFFE2

b) $s6 = 256
$t0 = 256 + 4 = 260
$t1 = 256 + 0 = 256

instruções → 0x00000104 = 0x00000100 ← dado

$t0 = 256
$s0 = 256 + 256 = 512 → $s0 = 0x00000200

2.10)

a) 000000 10000 10000 10000 00000 100000

2.10.1) ADD $s0, $s0, $s0 ← tipo R

2.10.2) Instrução tipo R

2.10.3) 0x02108020

2.5.2) # insertion sort no MIPS

```
        ADD $s0, $0, $0
LOOP1:
        SLTi $t0, $s0, 4
        BEQ $t0, $0, EXIT1
        ADDI $s1, $s0, 1
LOOP2:
        SLTi $t0, $s1, 5
        BEQ $t0, $0, EXIT2
        SLL $t0, $s1, 2
        ADD $t0, $t0, $s6
        LW $t1, 0($t0)
        SLL $t2, $s1, 2
        ADD $t2, $t2, $s6
        LW $t3, 0($t2)
        BLT $t1, $t3, EXIT3
        SW $t1, 0($t2)
        SW $t3, 0($t0)
EXIT3:
        ADDi $s1, $s1, 1
        J LOOP2
EXIT2:
        ADDi $s0, $s0, 1
        J LOOP1
EXIT1:
        ...
```

variável i e loop1 para percorrer o vetor

variável j e loop2 para percorrer o vetor

pega o valor do vetor na posição vetor[i]

pega o valor do vetor na posição vetor[j]

verifica se vetor[i] < vetor[j]

swap caso vetor[j] seja menor

itera j e volta pro loop 2

itera i e volta pro loop 1