5.22

	ADDCC %r10, -100, %r15			
	T1	T2	T3	T4
IEM_WE	0	0	0	0
DR_FOR_DATA	0	X	X	X
R_WE	1	0	0	0
_ALL	X	X	X	0
U_IMM	X	1	X	X
ER_WE	0	1	0	0
SUL_WE	0	0	1	0
_FROM_PC	X	X	X	0
_FOR_SETHI	X	X	X	0
_FROM_MEM	X	X	X	0
GIS_WE	0	0	0	1

5.5

4 ciclos

① [Instr.] <= MEM[pc] ; [pc] <= [pc] + 4

② [A] <= [rs] ; [B] <= [rt]

③ [rt] <= [A]

④ [rs] <= [B]

swap $0, $X, $y

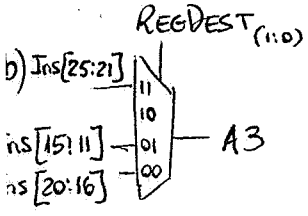
swap $X, $y

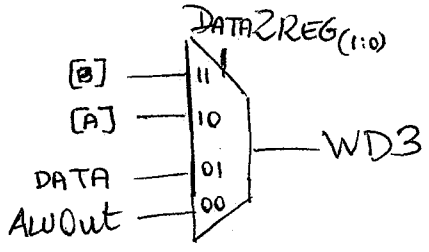
a)

6b(op)	rs	rt	rd	shamt	6bit(funct)
000000	01111	10001	00000	00000	110000

x = 15
y = 17
} por ejemplo

b)





5.17

b)

	T-1	T	T+1
PC	0x12C	0x00C	0x010

c) 0x00C ⇒ %ra

Etiqueta R ⇒ 0x190

d)

0x03E00008 ⇒ 0000 0011 1110 0000 0000 0000 0000 1000

r31 r0

↓ ↓

rs rt

ciclo 2

rs = r31 %ra = 0x11C

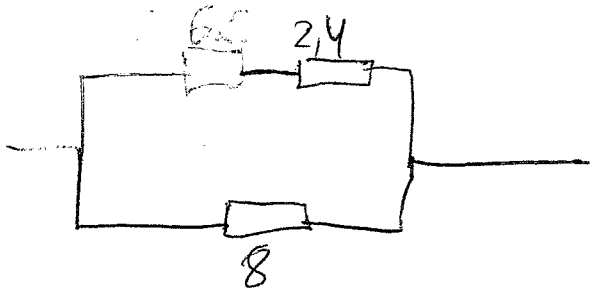
rt = r0 0

ciclo 3

rs = r31 %ra = 0x11C

rt = r0 0

	MemToReg	MemWrite	Branch	ALUControl	ALUSrc	RegDest	RegWrite	RegToPC	ExtCero	Jump	PCtoReg
R-Type	0	0	0	?	0	1	1	0	x	0	0
Lw	1	0	0	010 (+)	1	0	1	0	0	0	0
Sw	x	1	0	010 (+)	1	x	0	0	0	0	x
Beq	x	0	1	110 (-)	0	x	0	0	x	0	x
ógicas Inm	0	0	0	?	1	0	1	0	1	0	0
itm. Inm	0	0	0	?	1	0	1	0	0	0	0
J	x	0	x	x	x	x	0	0	x	1	x
Jal	x	0	x	x	x	x	1	0	x	1	1
Jr	x	0	x	x	x	x	0	1	x	0	x



4,098 Ω

$I = V/R = 17/4{,}098 = 2{,}9\ V$

$I_{67} = I_{567} = I_5 = \frac{V}{R_{567}} = \frac{17\ V}{8\ \Omega} = 1{,}5\ A$

$V_{67} = V_6 = V_7 = I_{67} \cdot R_{67} = 1{,}5\ A \cdot 4\ \Omega = 6\ V$

$I_6 = \frac{V_6}{R_6} = 0{,}75\ A$

$I_7 = \frac{V_7}{R_7} = 0{,}75\ A$

$I_{1234} = I - I_{567} = 1{,}4\ A = I_1 = I_{234}$

$V_{234} = I_{234} \cdot R_{234} = 1{,}4\ A \cdot 2{,}4\ \Omega = 3{,}36\ V = V_2 = V_{34}$

$I_2 = \frac{V_2}{R_2} = \frac{3{,}36\ V}{4\ \Omega} = 0{,}84\ A$

$I_{34} = I_3 = I_4 = I_{234} - I_2 = 0{,}56\ A$

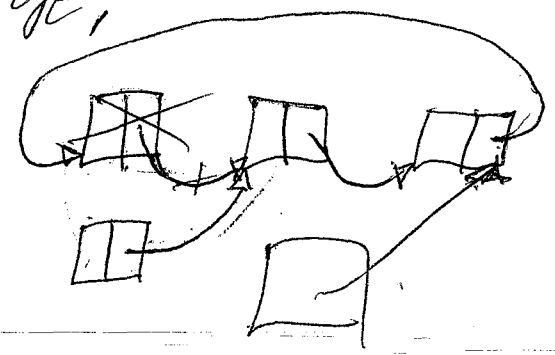
~~Node~~ * Element * info;

for (pn = FIRST(pl); NEXT(NEXT(pn)) != NULL; pn = NEXT(pn));

ele_copy(info = pn -> next -> info);

node_destroy(NEXT(pn)->info);

return info;



(listaEC *pl)

Ele * ele;

Nodo * aux;

aux = node_init();

aux = NEXT(FIRST(pl));

ele = node_get_info(ele);

FIRST(pl)

node_destroy(FIRST(pl));

NEXT(LAST(pl)) = NEXT(aux);

node_destroy(aux);

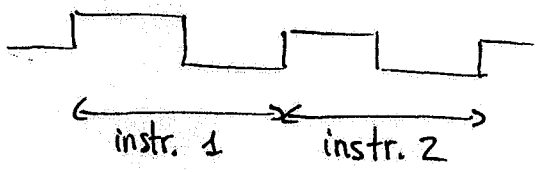
return ele;

TEMA 4 EL PROCESADOR II: DISEÑO Y CONTROL DE

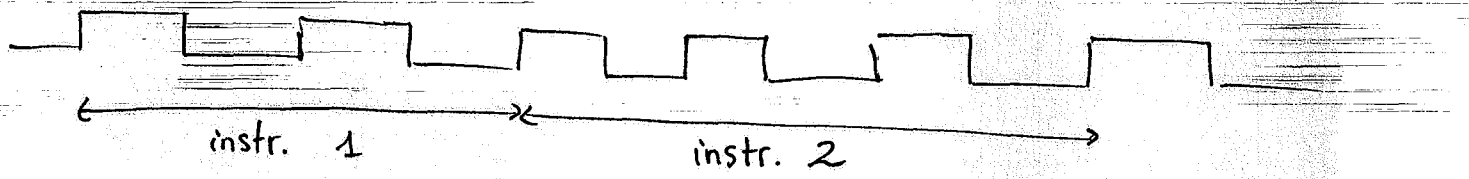
IMPLEMENTACIÓN HARDWARE DE LA ARQUITECTURA (MICROARQUITECTURA

→ Uniciclo: cada instrucción se ejecuta en un ciclo
→ Multiciclo: instrucción se divide en pasos cortos
→ Segmentado (pipelined)

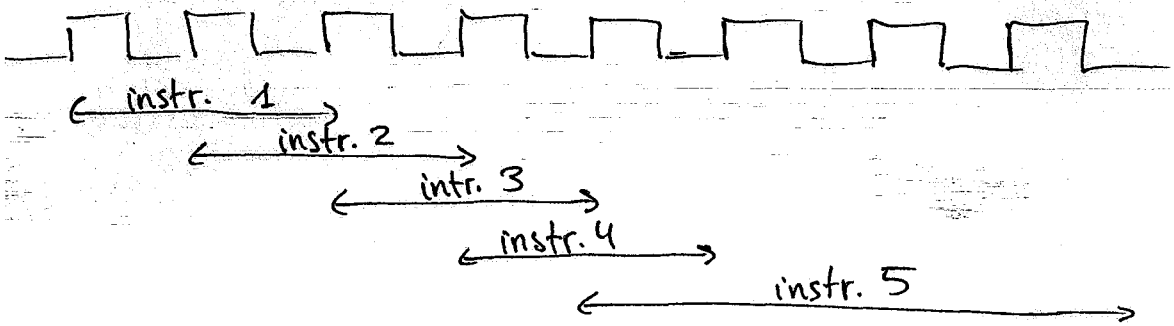
UNICICLO (U4)



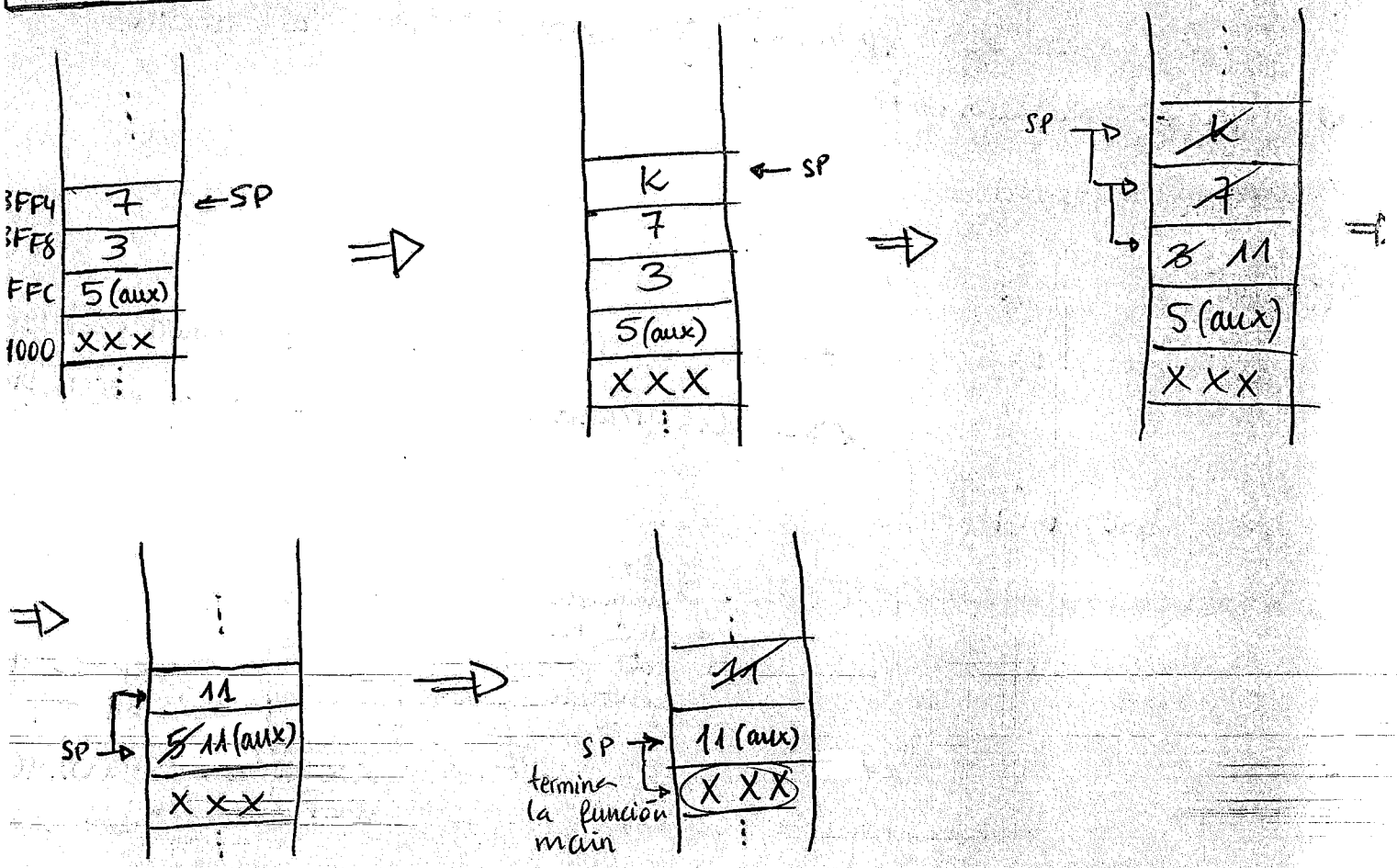
MULTICICLO (U5)



SEGMENTADO (3er curso)



PROBLEMA 3.17



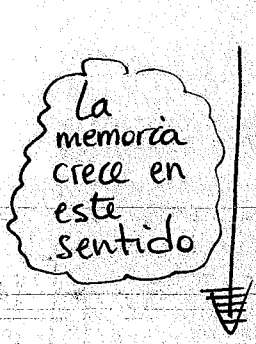
PILA DEL SISTEMA

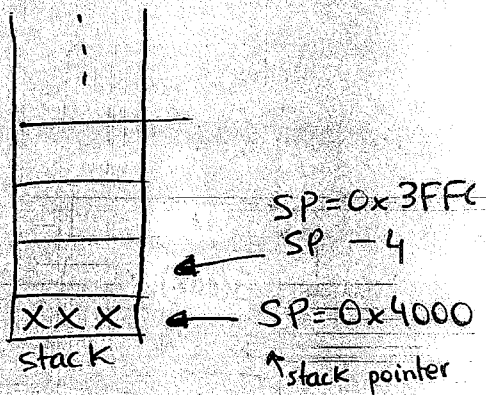
La pila sirve para preservar los contenidos de los registros ados en el procedimiento

$s0 = result

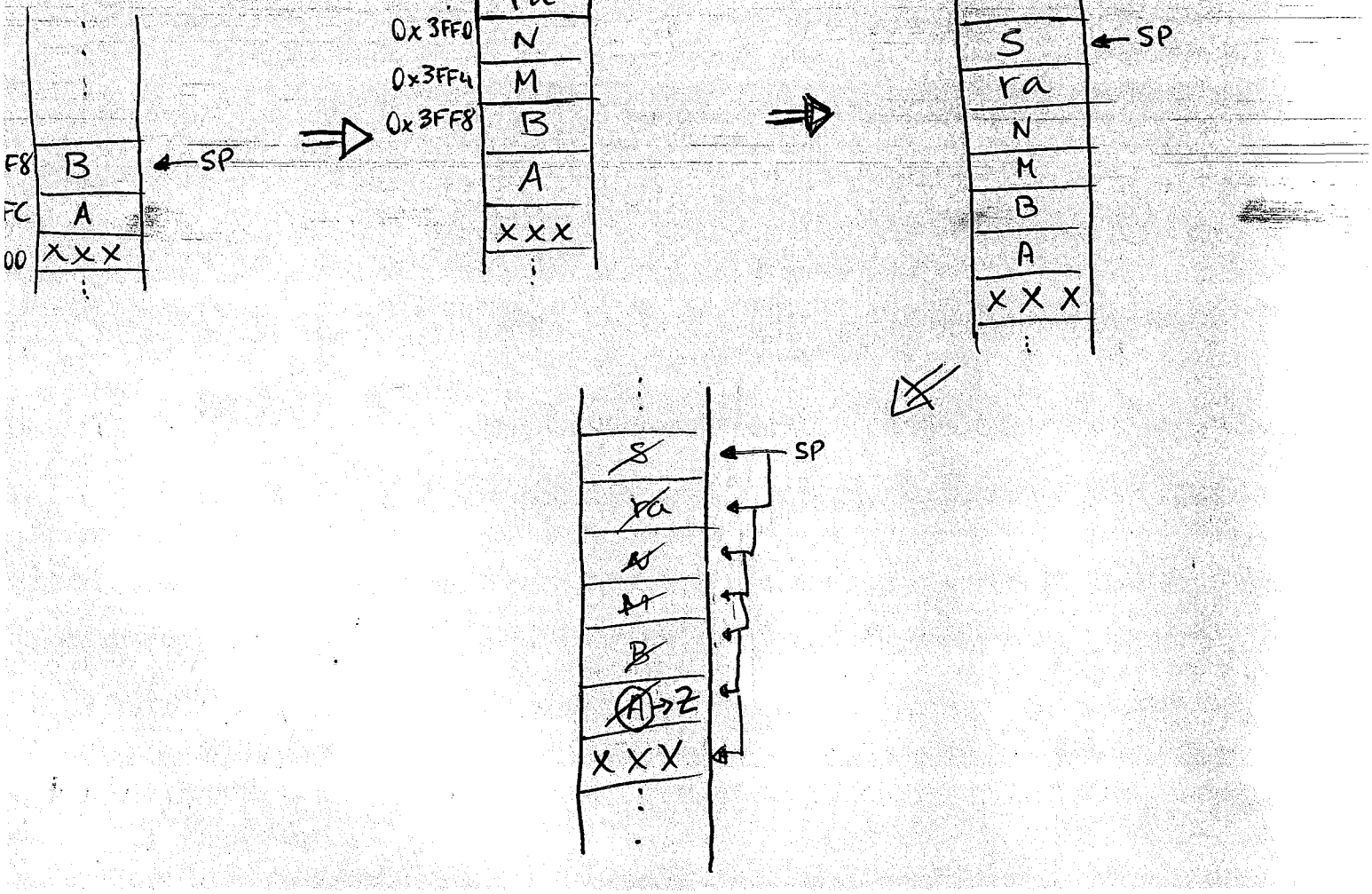
iff of sums

```
addi $sp, $sp, -4    # reserva espacio en la pila para guardar el registr
                     # $s0. Los registros $t0 y $t1 no se pueden preser
sw   $s0, 0($sp)
```





PROBLEMA 3.5



Dado el siguiente programa escrito para MIPS, en donde [illegible] instrucciones en ensamblador y en código máquina (hexadecimal).

Se pide:

a) Complete el código ensamblador.

b) Señale la tabla de ~~signos~~ símbolos.

c) Indique la función que ejecuta el código y escriba el valor de la posición de memoria señalada por la etiqueta E

x8C112000 ⟶ OP | rs | rt

1000 11|00 000|1 0001|0010 0000.

LW $17, $0

2232FFFF ⟶ OP | rs | rt

0010 00|10 001|1 0010|1111 1111 1111 1111

ADDI 17 18

ADDI $S2, $S1, -1

3.11.

64 registros ⇒ 6 bits

4 bits para dif. instr en TIPO 1

TIPO 1 | OP1 | OP2 | RegFuente1 | RegFuente2 | RegTarget

2b | 4b | 6b | 6b | 6b

14 instr.; 3Regs ↳ cada 1 de 6

TIPO 2 | OP1 | instrucción | SIN USO | RegFuente | RegDestino

2b | 6 bit | 4bit | 6 bit | 6bit

47 instr., 2 regs, ↳ 6 bit cada uno

↳ 6 bit

TIPO 3 | OP1 | instrucción | NO USO | RegDestino

2b | 2 bit | 14 bit | 6b

4 instr., 1 regs,

↳ 2bit 6 bit

24 bits

65 instrucciones diferentes ⇒ 7 bits

2 bits para diferenciar tipo

Direcciones de memoria posibles ⇒ 2^{14}

¿Se puede mejorar?

Sí ⟶ con 00 → TIPO 1

con 01 → TIPO 2

con 1 → TIPO 3 ⇒ OP1 instr | NO USO

1b | 2bit | 15 | 6b

2^{15} que es el doble de 2^{14}.

3.9 $P = M * N$ donde $N = 2^X - 2^Y$

M, X e Y en memoria, con $X, Y \in [0, 31]$ (positivos de 0 a 31)

```
.text 0x0000
    lw   $t1, M($0)
    lw   $t2, X($0)
    lw   $t3, Y($0)
    sllv $t4, $t1, $t2
    sllv $t5, $t1, $t3
    sub  $s1, $t4, $t5
lo da el enunciado
    sw   $s1, P($0)
fin: j fin

.data 0x2000
M:
X:
Y:
P:
```

3.19

a) `sw $v0, 0x2008($0)`

op	rs	rt	imm
101011	00000	00010	0010 0000 0000 1000

0xAC022008

b) `lui $t4, $0, 0x1234` → pone 0x12340000

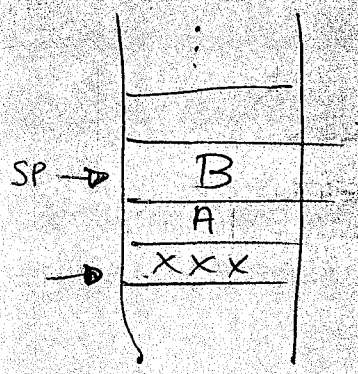
`ori $t4, $t4, 0x5678` → 0x12340000 or 0x00005678 = 0x12345678

c)
```
lw   $s1, A($0)
sub  $s1, $0, $s1   ← TRUCO COMPLEMENTO A DOS
addi $t0, $0, 4
sw   $s1, A($t0)
```

18]

a) ↓ ILA

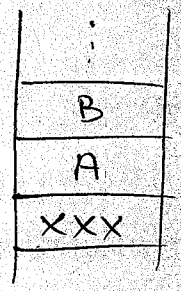
```
main: lw   $s0, A ($0)
      lw   $s1, B ($0)
      addi $sp, $sp, -8
      sw   $s0, 4 ($sp)
      sw   $s1, 0 ($sp)
      jal  SumMed
```

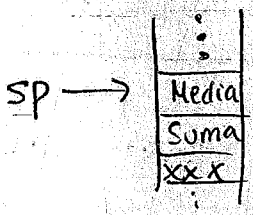


...

```
SumMed: lw   $t1, 0 ($sp)    # $t1 ⇐ B
        lw   $t2, 4 ($sp)    # t2 ⇐ A
        add  $t1, $t1, $t2   # t1 ⇐ t1 + t2
        sra  $t0, $t1, 1     # to ⇐ t1/2
        sw   $t0, 0 ($sp)
        sw   $t1, 4 ($sp)
        jr   $ra
```

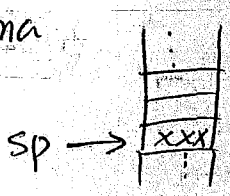
con signo ⇐ sra





...

```
# main de vuela
lw $s1, 0 ($sp)   # Media
lw $s2, 4 ($sp)   # Suma
addi $sp, $sp, 8
```



b)

↓

EGISTROS
PECÍFICOS

```
main:   lw   $a0, A($0)   # $a0 ⇐ A
        lw   $a1, B($0)   # $a1 ⇐ B
        jal  SumMed

        ...

SumMed:
        addi $v0, $a0, $a1   # $v0 ⇐ A+B
        sra  $v1, $v0, 1     # $v1 ⇐ (A+B)/2
        jr   $ra

# de vuelta en el main
        sw   $v0, A($0)
        sw   $v1, B($0)
```

3.16

```
.text  0x0100
med4:  add  $t0, $a0, $a1
       add  $t1, $a2, $a3
       add  $t0, $t0, $t1
       sra  $v0, $t0, 2
       jr   $ra
```

$ra → 0x0014

$ra : 0x0014
R: 6
A: 0x2000
MED4 = 0x0100
FIN = 0x0018

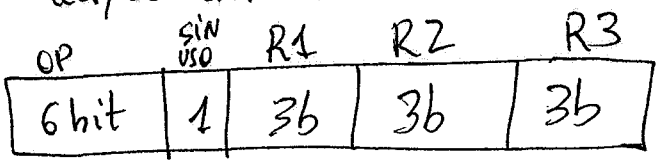
:.12| Arquitectura de 16 bits

6 registros ⇒ 3 bits

Espacio de direccionamiento de 64Kbytes ⇒ 2^{16} ⇒ 16 bits

- Todos los códigos de operación la misma longitud
- No todas las instrucciones tienen el mismo tamaño.
- Se puede leer/escribir de memoria un solo byte.

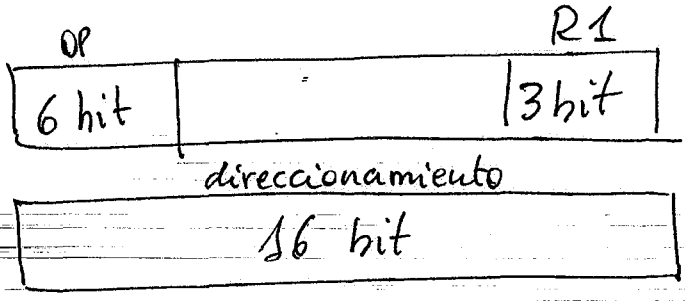
TIPO 1



40 instr.

3 Registros ⇒ 9 bits (3 cada uno)

TIPO 2

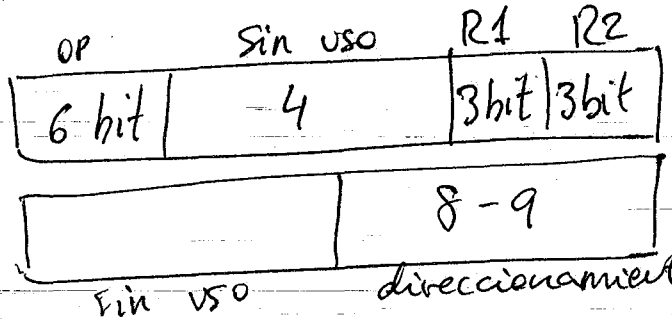


5 instr.

16 bit direccion. de memoria

1 R ⇒ 3 bits

TIPO 3

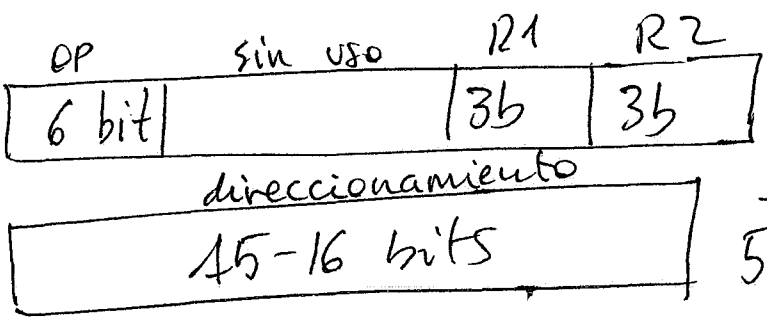


5 instr.

2 Registros ⇒ 6 bit

9-8 bits de direccionamiento

TIPO 4



4 instr.

54 instr.

⇓

6 bit (OP)

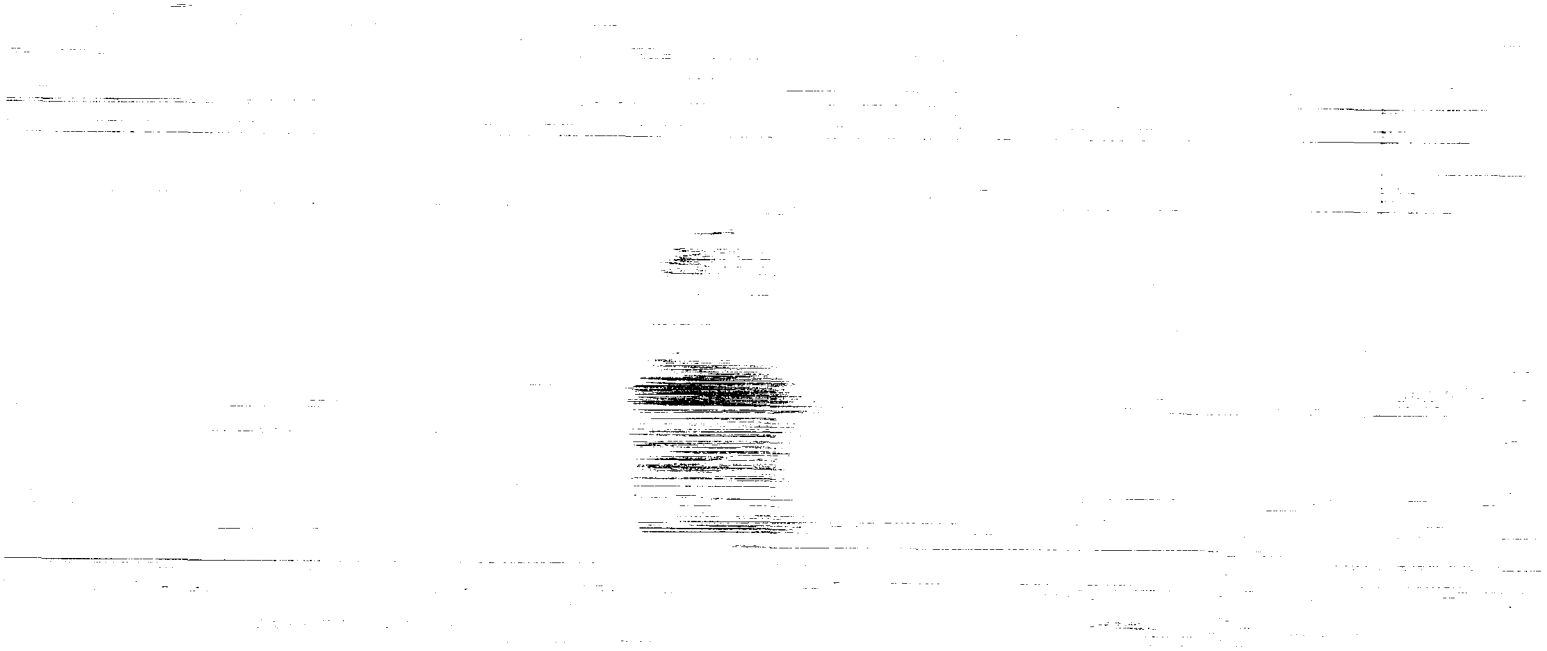
≥ 15 bits

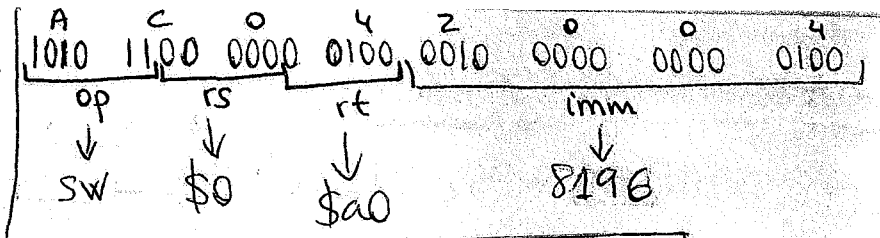
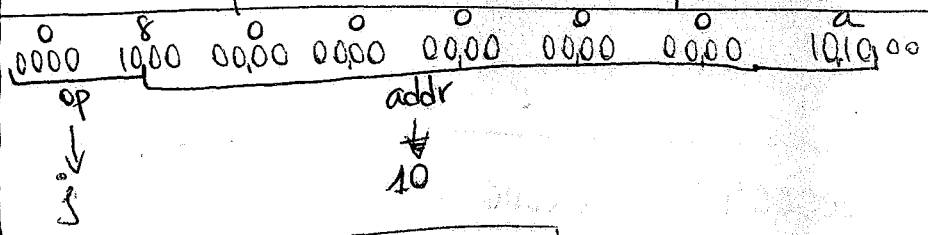
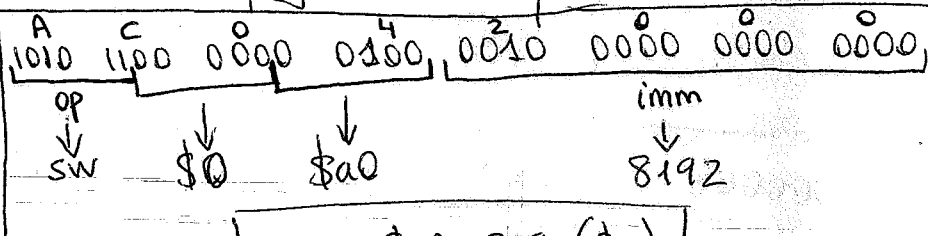
2 Reg ⇓ 6 bit

R-Type: todos los operandos están en registros.

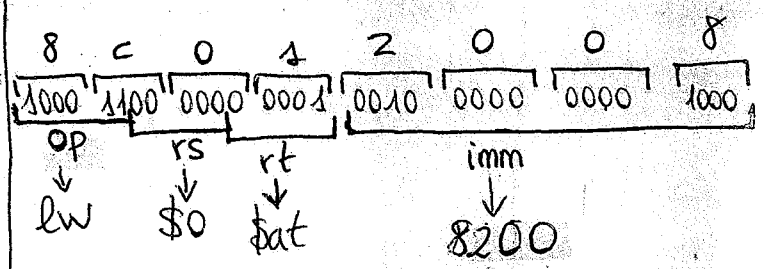
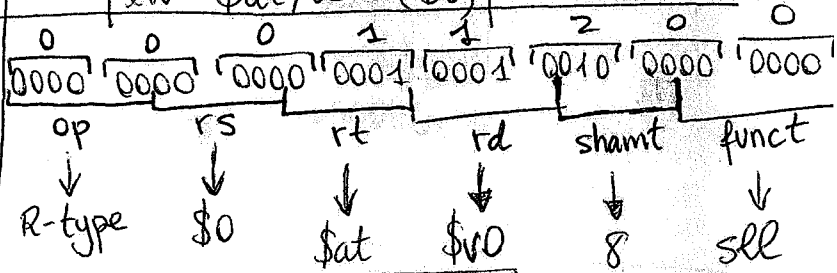
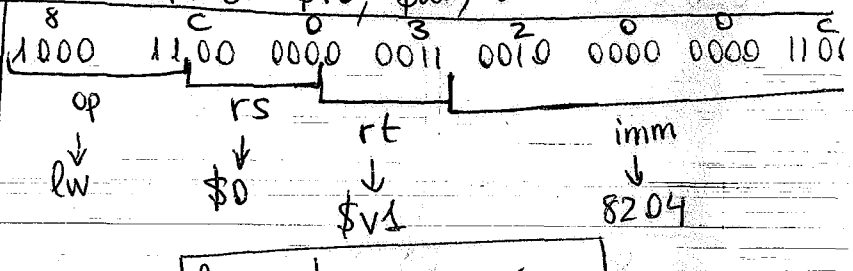
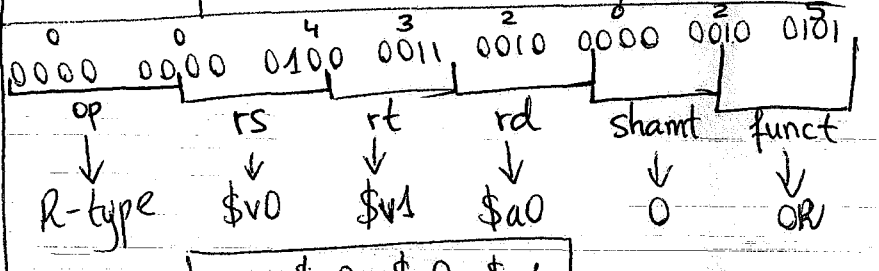
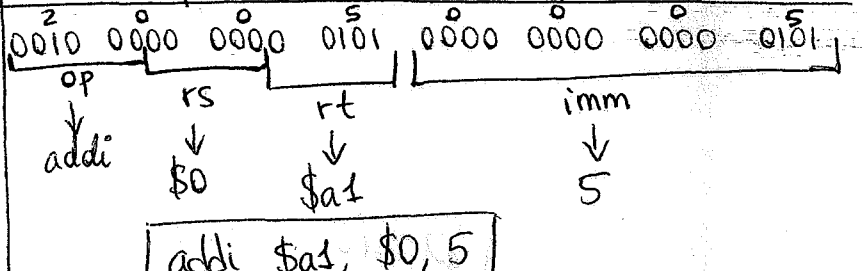
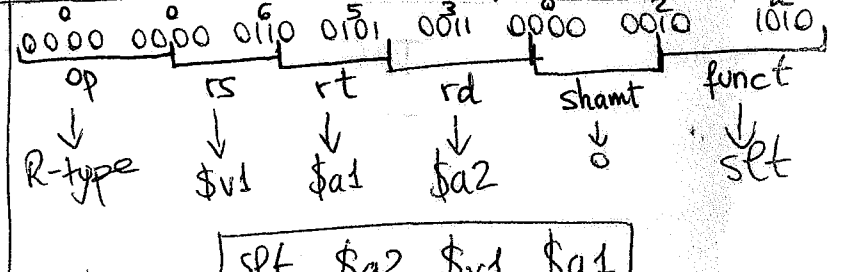
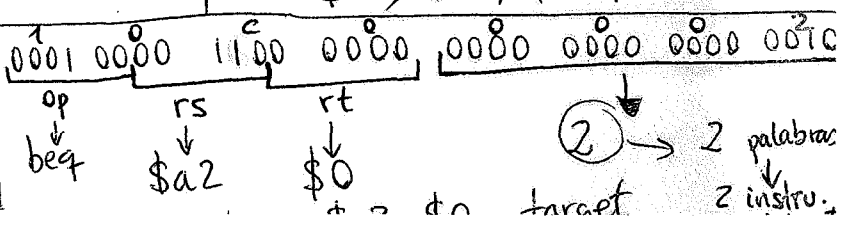
I-Type: aparte de registros hay un operando inmediato (constante).

J-type: instrucciones usadas para saltos.



x0000001C	0xac042004	 sw $a0, 8196($0)
x00000020	0x0800000a target2 = =0x00000028	 j target2
x00000024	0x ac042000	 sw $a0, 8192($0)
00000028	0x0800000a	Igual al 0x00000020 j target3 target3 = 0x00000028 target2 = target3

FOLIO DEL SEGMENTO DEL CÓDIGO

Address	Code	
x00000000	0x8c012008	 lw $at, 8200($0)
x00000004	0x00011200	 sll $v0, $at, 8
x00000008	0x8c03200c	 lw $v1, 8204($0)
x0000000c	0x00432025	 or $a0, $v0, $v1
x00000010	0x20050005	 addi $a1, $0, 5
x00000014	0x0065302a	 slt $a2, $v1, $a1
00000018	0x10c00002	 target: 0x00000024

SEGMENTO DE DATOS

Address	Code	
0002000	0x00000000	→ 0
0002004	0x00000000	→ 0
00002008	0x0000ceba	→ 0000... 1100 1110 1011 1010 → 52922
0000200C	0x00000005	→ 5

2048
1024
512
128
32
16
8
2
16384
32768

+ 0 = 52922