$a+0=a$ | $a\cdot 1=a$

$a+bc=(a+b)(a+c)$ | $a(b+c)=ab+ac$

$a+\bar{a}=1$ | $a\cdot\bar{a}=0$

$a+1=1$ | $a\cdot 0=0$

$a+ab=a$ | $a(a+b)=a$

$a+\bar{a}b=a+b$ | $a(\bar{a}+b)=ab$

$ab+a\bar{b}=a$ | $(a+b)(a+\bar{b})=a$

$ab+a\bar{b}c=ab+ac$ | $(a+b)(a+\bar{b}+c)=(a+b)(a+c)$

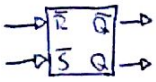
$ab+\bar{a}c+bc=ab+\bar{a}c$ | $(a+b)(\bar{a}+c)(b+c)=(a+b)(\bar{a}+c)$

$NAND(A,A)=\bar{A}=NOR(A,A)$

$NAND(NAND(A,B),NAND(A,B))=A\wedge B=NOR(NOR(A,A),NOR(B,B))$

$NAND(NAND(A,A),NAND(B,B))=A\vee B=NOR(NOR(A,B),NOR(A,B))$

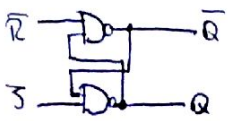
R/S



$\bar{R}=\bar{S}=1\Rightarrow Q=cte$

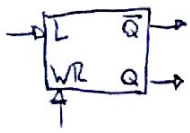
$\bar{R}=0\Rightarrow Q=0$

$\bar{S}=0\Rightarrow Q=1$



Registro Master/Slave

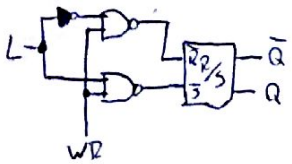
Latch

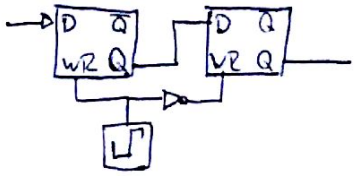


$WR=1\Rightarrow Q=L$

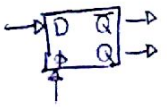
$WR=0\Rightarrow Q=L_{t-1}$

Q vale L antes del cambio

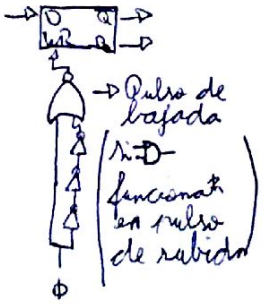




Flip-Flop Data



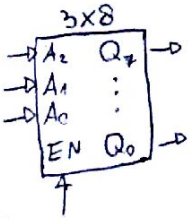
Q = el valor de D en el pulso de bajada (antes de emp.)



Latch no sirve para circ. recuenciales

Decodificador

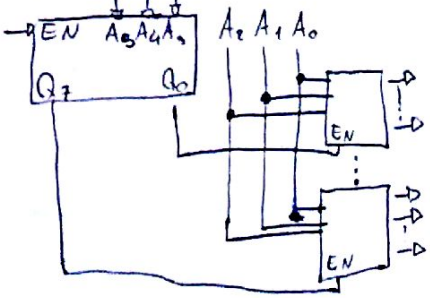
3x8



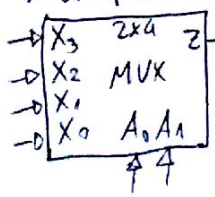
$EN=0\Rightarrow Q_i=0$

$EN=1\Rightarrow Q_{Ai,Ao}=1$

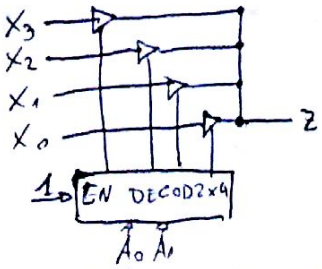
Decodificador 6x64



Multiplexor

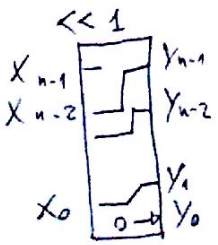


$Z=X_{Ai,Ao}$



Circ. sec: #finito estados & Estado cte. durante ciclo reloj & salidas = f(entradas, Q - est. int.) & al final calc. est. sig. | Circ. Rec. min: Flip-flop data

C



b-8bit, w-16bit, l=32bit
add(adc,sbb) mul s
sub:= d=d-s imul s
inc:=d++ div s
dec:=d-- idiv s
neg:=d*-1

mul	arg2	destH	destL
8	AL	AH	AL
16	AX	DX	AX
32	EAX	EDX	EAX

div	dividend	remain	quot
8	Ax	AH	AL
16	DX, Ax	DX	AX
32	EDX, EAX	EDX	EAX

disp(base,index,scale)

$: numero %reg
mov s,d
lea mem,reg load address of mem into reg
movs s,d extend w sign
movz s,d " w 0's
cmp a1,a2 # sub a1,a2
test a1,a2 # bitwise AND

je	on equal	jnz	on not zero
jne	on not equal	jz	on zero
jg	on greater	ja	signed
jge	on = equal	jae	"
jl	on less	jb	"
jle	on less equal	jbe	"
jo	on overflow	jmp	jump

and s,d	shr d	>>1
or s,d	shl d	<<1
xor s,d	sar d	>>1 w/sign
not d	sal d	<<1
ror d	scr arg	
rol d	scl arg	

```
q
cmp a,b
jg L1  # if(a>b)
test 1,b
jnz L2  # if(b(lowght)==1)
test a,a
jz label #if(a=0)
cmp a,b
jz L5  # if(a==b)
```

```
push s
pop d # addl x,esp / movl x(%esp),d
call lab
ret
```

1 Llamador apila args en orden inverso y llama con call
2 Se encadenan reg act. ebp
3 Se reg.-reg segun convención (ebx, esi, edi, esp, ebp. modificar: eax, ecx, edx
4 Se agranda reg act. para variables locales
5 calculos
6 valor de ret en eax
7 botar variables globales y restituir reg guardados
8 usar ret
9 desapilar args

```
res=proc(arg1, arg2)

push arg2
push arg1
call proc
movl %eax,res
addl $8,%esp
```

```
int proc(int p1, int p2){
  int local;
  ...
  return local;
}

proc:
  pushl %ebp
  movl %esp,%ebp
  pushl edi
  pushl esi
  pushl ebx
  subl 4,esp
  ...
  movl -16(%ebp), eax
  addl 4, esp
  popl ebx
  popl esi
  popl edi
  leave
```

```
leave
movl ebp,esp
popl ebp
```

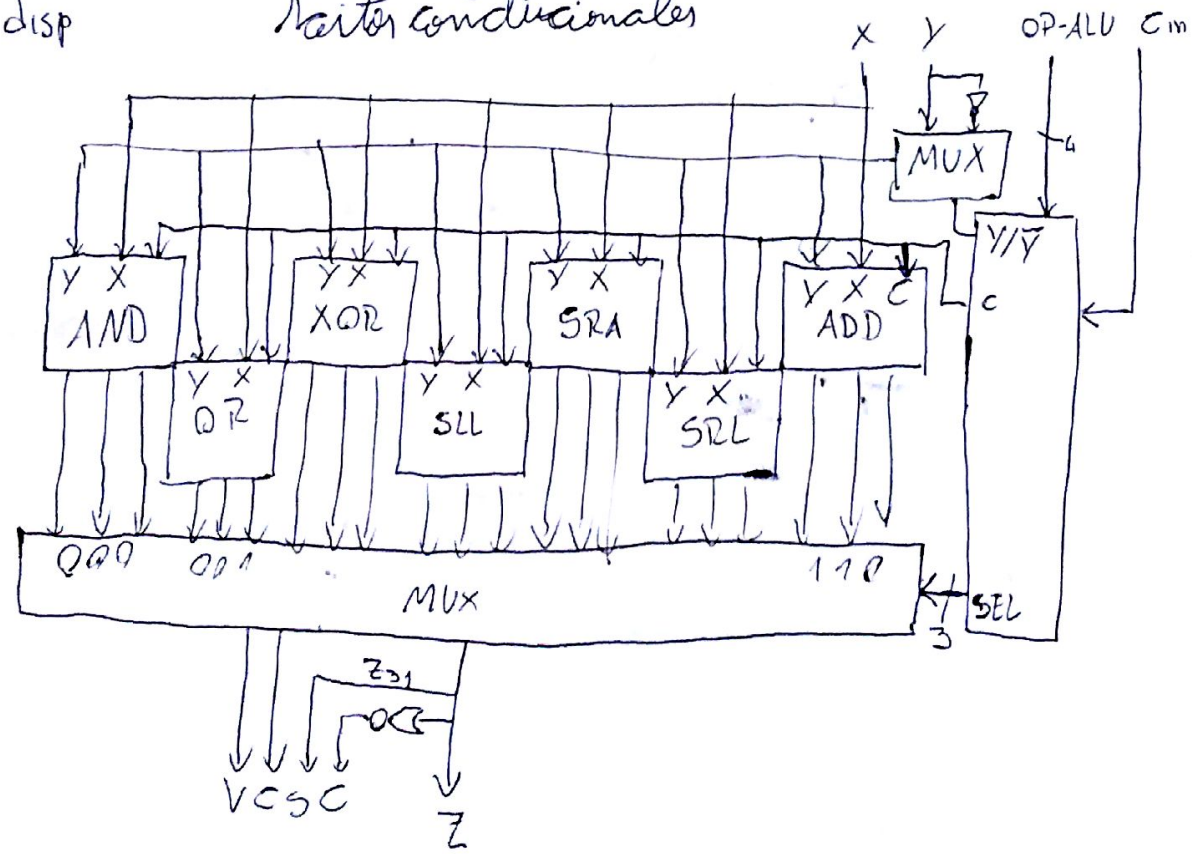
AR: Address Register
IR: Instruction register
PC: Program Counter
R-BANK: Registry bank
SR: Flags (Validez 1 ciclo de reloj)

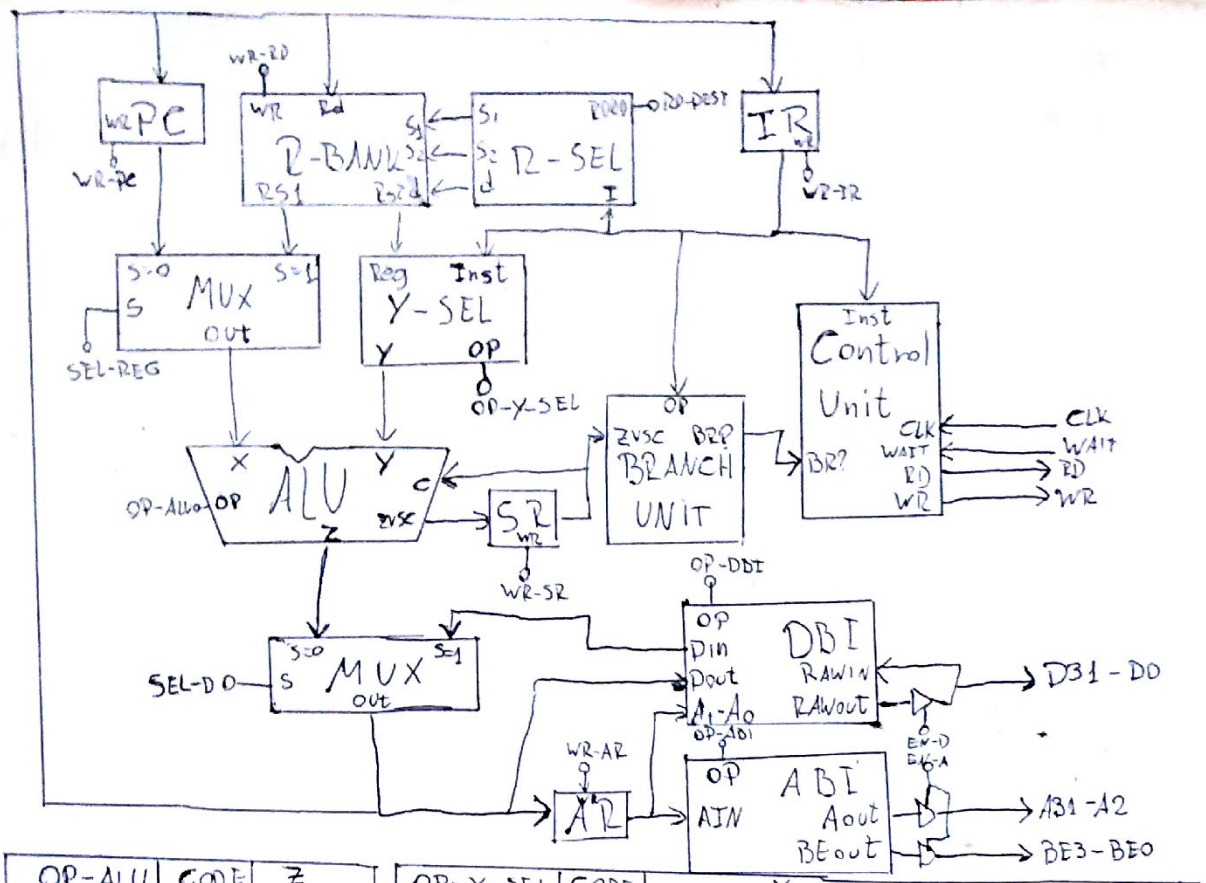
ABI: Address bus interface
DBI: Data bus interface

Fetch1: Pido la instrucción a ejecutar (PC en bus de direcciones y Read)
Fetch2: Escribir la instrucción en IR
Decode: PC++ (@4→ALU) mientras Control unit decodifica
Execute: Ejecuta

M32: arquitectura lógica

op reg_s, val, $regd$	$regd = reg_s$ op val
op addr, reg	Lectura en memoria
op reg, addr	Escritura en memoria
salto disp	Saltos condicionales





OP-ALU	CODE	Z
@ADD	0000	X+Y
@ADDX	0001	X+Y+C
@SUB	0010	X+~Y+1
@SUBX	0011	X+~Y+C
@AND	0100	X&Y
@OR	0101	X|Y
@XOR	0110	X xor Y
@SLL	0111	X<<Y
@SRL	1000	X>>Y
@SRA	1001	X>>sY

OP-Y-SEL	CODE	Y
@0	00	0
@4	01	4
@INST	10	if(INST[13]=1)Exts(INST[12-0])else Reg
@DISP	11	Exts(INST[23-0])

OP-ABI	CODE	Descripcion
@W	00	Direcciona una palabra (word)
@H	01	" media " (Half-word)
@B	10	" un byte

M32: Arquitectura

OP-DBI	CODE	Descripcion
@LDW	000	Lee una palabra (word)
@LDUH	001	" media " sin signo
@LDSH	010	" " " con "
@LDUB	011	" un byte sin "
@LDSB	100	" " " con "
@STW	101	Escribe una palabra
@STH	110	" media "
@STB	111	" " "