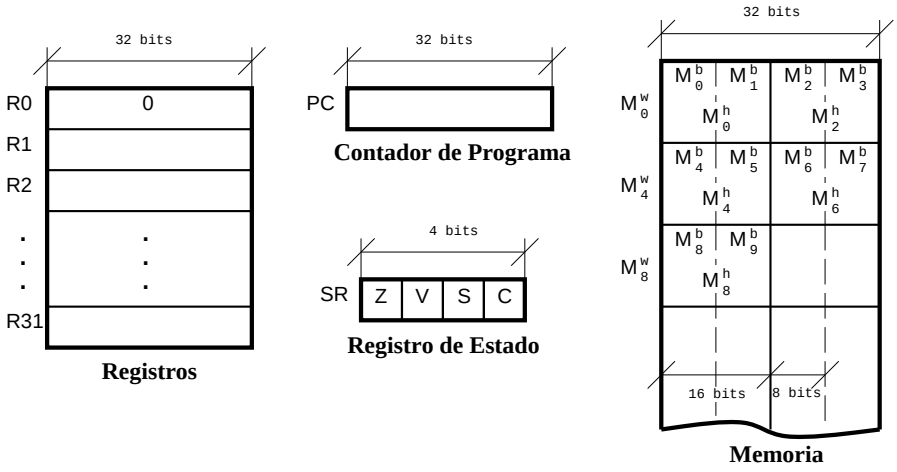
M32: Arquitectura Lógica

M32 es una CPU diseñada sólo con fines docentes y por lo tanto no posee todas las capacidades exigidas a una CPU real. La arquitectura de registros de M32 es la siguiente:



Nótese que una misma dirección en memoria puede ser vista como un *byte*, como parte de una media palabra o como parte de una palabra completa.

Las instrucciones *assembler* de M32 son de la forma:

Operador	Operandos	Descripción
`op`	reg_s, *val*, reg_d	operaciones del tipo $reg_d = reg_s$ `op` *val*
`op`	*addr*, *reg*	lectura en memoria
`op`	*reg*, *addr*	escritura en memoria
`salto`	*disp*	saltos condicionales

En donde cada uno de los operandos corresponde a:

reg	Cualquier registro entre $\%\mathtt{R}_0$, $\%\mathtt{R}_1$ ... $\%\mathtt{R}_{31}$
imm	Un valor binario $\in [-2^{12}, 2^{12}-1]$ (representable en 13 bits)
addr	Una dirección de la forma $[reg' + reg'']$ o $[reg + imm]$
val	Un valor de la forma *reg* o *imm*
disp	Un desplazamiento $\in [-2^{23}, 2^{23}-1]$ (representable en 24 bits)

Definiciones	
$Rep_s^n(x)$	Representación en n bits con signo del entero x
$Rep_u^n(x)$	Representación en n bits sin signo del entero positivo x
Ext_0(x)	Conversión de x a 32 bits extendiendo con ceros
Ext_s(x)	Conversión de x a 32 bits extendiendo con el signo
$Trunc_b$(x)	Trunca la palabra x a 8 bits
$Trunc_h$(x)	Trunca la palabra x a 16 bits

La función $\langle x \rangle_t$ que aparece en la segunda columna se define como:

$$\langle x \rangle_t = \textit{el valor del registro/memoria } x \textit{ en } t$$
$$\langle [reg' + reg''] \rangle_t = \langle reg' \rangle_t \oplus \langle reg'' \rangle_t$$
$$\langle imm \rangle_t = Ext_s(Rep_s^{13}(imm))$$
$$\langle [reg + imm] \rangle_t = \langle reg \rangle_t \oplus Ext_s(Rep_s^{13}(imm))$$

A continuación se describirá cada una de las posibles instrucciones assembler de M32.

Instrucciones de lectura en memoria		
Instrucción assembler	Operación	Tipo leido
`ldw` *addr*, *reg*	$\langle addr \rangle_t \& 3 \neq 0 \Rightarrow$ `TRAP` $reg \not\equiv \%\mathtt{R}_0 \Rightarrow \langle reg \rangle_{t+1} = \mathtt{M}^w_{\langle addr \rangle_t}$	una palabra
`lduh` *addr*, *reg*	$\langle addr \rangle_t \& 1 \neq 0 \Rightarrow$ `TRAP` $reg \not\equiv \%\mathtt{R}_0 \Rightarrow \langle reg \rangle_{t+1} = Ext_0(\langle \mathtt{M}^h_{\langle addr \rangle_t} \rangle_t)$	media palabra sin signo
`ldub` *addr*, *reg*	$reg \not\equiv \%\mathtt{R}_0 \Rightarrow \langle reg \rangle_{t+1} = Ext_0(\langle \mathtt{M}^b_{\langle addr \rangle_t} \rangle_t)$	*byte* sin signo
`ldsh` *addr*, *reg*	$\langle addr \rangle_t \& 1 \neq 0 \Rightarrow$ `TRAP` $reg \not\equiv \%\mathtt{R}_0 \Rightarrow \langle reg \rangle_{t+1} = Ext_s(\langle \mathtt{M}^h_{\langle addr \rangle_t} \rangle_t)$	media palabra con signo
`ldsb` *addr*, *reg*	$reg \not\equiv \%\mathtt{R}_0 \Rightarrow \langle reg \rangle_{t+1} = Ext_s(\langle \mathtt{M}^b_{\langle addr \rangle_t} \rangle_t)$	*byte* con signo
Instrucciones de escritura en memoria		
Instrucción assembler	Operación	Tipo escrito
`stw` *reg*, *addr*	$\langle addr \rangle_t \& 3 \neq 0 \Rightarrow$ `TRAP` $\langle \mathtt{M}^w_{\langle addr \rangle_t} \rangle_{t+1} = \langle reg \rangle_t$	una palabra
`sth` *reg*, *addr*	$\langle addr \rangle_t \& 1 \neq 0 \Rightarrow$ `TRAP` $\langle \mathtt{M}^h_{\langle addr \rangle_t} \rangle_{t+1} = Trunc_h(\langle reg \rangle_t)$	media palabra
`stb` *reg*, *addr*	$\langle \mathtt{M}^b_{\langle addr \rangle_t} \rangle_{t+1} = Trunc_b(\langle reg \rangle_t)$	*byte*

Instrucciones aritméticas		
Instrucción assembler		Operación
`add`	reg_s, val, reg_d	$reg_d \not\equiv \%\mathtt{R}_0 \Rightarrow \langle reg_d \rangle_{t+1} = \langle reg_s \rangle_t \oplus \langle val \rangle_t$ $\langle v \rangle_{t+1} = Ovf(\langle reg_s \rangle_t, \langle val \rangle_t, 0)$ $\langle c \rangle_{t+1} = Carry(\langle reg_s \rangle_t, \langle val \rangle_t, 0)$
`addx`	reg_s, val, reg_d	$reg_d \not\equiv \%\mathtt{R}_0 \Rightarrow \langle reg_d \rangle_{t+1} = \langle reg_s \rangle_t \oplus \langle val \rangle_t \oplus \langle c \rangle_t$ $\langle v \rangle_{t+1} = Ovf(\langle reg_s \rangle_t, \langle val \rangle_t, \langle c \rangle_t)$ $\langle c \rangle_{t+1} = Carry(\langle reg_s \rangle_t, \langle val \rangle_t, \langle c \rangle_t)$
`sub`	reg_s, val, reg_d	$reg_d \not\equiv \%\mathtt{R}_0 \Rightarrow \langle reg_d \rangle_{t+1} = \langle reg_s \rangle_t \oplus \sim \langle val \rangle_t \oplus 1$ $\langle v \rangle_{t+1} = Ovf(\langle reg_s \rangle_t, \sim\langle val \rangle_t, 1)$ $\langle c \rangle_{t+1} = Carry(\langle reg_s \rangle_t, \sim\langle val \rangle_t, 1)$
`subx`	reg_s, val, reg_d	$reg_d \not\equiv \%\mathtt{R}_0 \Rightarrow \langle reg_d \rangle_{t+1} = \langle reg_s \rangle_t \oplus \sim \langle val \rangle_t \oplus \langle c \rangle_t$ $\langle v \rangle_{t+1} = Ovf(\langle reg_s \rangle_t, \sim\langle val \rangle_t, \langle c \rangle_t)$ $\langle c \rangle_{t+1} = Carry(\langle reg_s \rangle_t, \sim\langle val \rangle_t, \langle c \rangle_t)$
Instrucciones lógicas		
`and`	reg_s, val, reg_d	$reg_d \not\equiv \%\mathtt{R}_0 \Rightarrow \langle \%\mathtt{R}_d \rangle_{t+1} = \langle reg_s \rangle_t \& \langle val \rangle_t$
`or`	reg_s, val, reg_d	$reg_d \not\equiv \%\mathtt{R}_0 \Rightarrow \langle \%\mathtt{R}_d \rangle_{t+1} = \langle reg_s \rangle_t \mid \langle val \rangle_t$
`xor`	reg_s, val, reg_d	$reg_d \not\equiv \%\mathtt{R}_0 \Rightarrow \langle \%\mathtt{R}_d \rangle_{t+1} = \langle reg_s \rangle_t \mathtt{xor} \langle val \rangle_t$
Instrucciones de desplazamiento		
`sll`	reg_s, val, reg_d	$reg_d \not\equiv \%\mathtt{R}_0 \Rightarrow \langle reg_d \rangle_{t+1} = \langle reg_s \rangle_t << \langle val \rangle_t$
`srl`	reg_s, val, reg_d	$reg_d \not\equiv \%\mathtt{R}_0 \Rightarrow \langle reg_d \rangle_{t+1} = \langle reg_s \rangle_t >> \langle val \rangle_t$
`sra`	reg_s, val, reg_d	$reg_d \not\equiv \%\mathtt{R}_0 \Rightarrow \langle reg_d \rangle_{t+1} = \langle reg_s \rangle_t >>_s \langle val \rangle_t$

Para todas las instrucciones aritméticas se cumple:

$$\langle z \rangle_{t+1} = \begin{cases} 1 & \text{Si} \langle reg_d \rangle_{t+1} = 0 \\ 0 & \text{sino} \end{cases}$$

$\langle v \rangle_{t+1} = 0$ (a menos que se indique otra cosa)

$\langle s \rangle_{t+1} = Sign(\langle reg_d \rangle_{t+1})$

$\langle c \rangle_{t+1} = \langle c \rangle_t$ (a menos que se indique otra cosa)

Instrucciones de Salto	
Instrucción assembler	Operación
`b`*cond* *disp*	*cond* se verifica $\Rightarrow \langle PC \rangle_{t+1} = \langle PC \rangle_t \oplus Ext_s(Rep_s^{22}(disp)) \oplus 4$ *cond* no se verifica $\Rightarrow \langle PC \rangle_{t+1} = \langle PC \rangle_t \oplus 4$
`jmpl` *addr*,*reg*	$\langle PC \rangle_{t+1} = \langle addr \rangle_t$ $reg \not\equiv$ %$\mathtt{R}_0 \Rightarrow \langle reg \rangle_{t+1} = \langle PC \rangle_t \oplus 4$

Condiciones de Saltos		
Código Instrucción	Tipo de salto	Condición de salto
ba	incondicional	1
be	igualdad	z
bne	desigualdad	$\overline{z}$
Comparaciones con signo		
bg	mayor	$\overline{z}(sv + \overline{s}\ \overline{v})$
bge	mayor o igual	$sv + \overline{s}\ \overline{v}$
bl	menor	$s\overline{v} + \overline{s}v$
ble	menor o igual	$s\overline{v} + \overline{s}v + z$
Comparaciones sin signo		
bgu	mayor	$c\overline{z}$
bgeu	mayor o igual	c
blu	menor	$\overline{c}$
bleu	menor o igual	$\overline{c} + z$