Trabajo de Estructura y Organización de Computadores

Diseño de una CPU

Nombre y Apellidos: Carlos Javier Hellín Asensio

Titulación: Ingeniería Informática Grupo: Tarde

Configuración de diseño:

RUTA DE DATOS: operadores

opción = 1 mod 5 = 1

RUTA DE DATOS: registro de datos

5 es impar → opción impar = 1

REPERTORIO DE INSTRUCCIONES: almacenamiento temporal

opción = 8 mod 5 = 3

REPERTORIO DE INSTRUCCIONES: soporte para subrutinas

opción = 4 mod 3 = 1

REPERTORIO DE INSTRUCCIONES: criterios de diseño del formato

3 es impar → opción impar = 1

UNIDAD DE CONTROL: microprogramación

opción = 4 mod 4 = 0

Repertorio de instrucciones

Se diseña el repertorio de instrucciones sabiendo que las operaciones de proceso admiten un operando ubicado en memoria, con el fin de adaptar la CPU a un uso de programas comerciales y a la configuración de los operadores que son: los lógicos (AND, OR, XOR, NOT), de desplazamiento a derecha e izquierda (SHR, SHL), de rotación a derecha e izquierda (ROR, ROL), sumador-restador de enteros (ADD, SUB) y de coma flotante (FADD, FSUB), multiplicador de enteros (MUL, IMUL) y coma flotante (FMUL) y, por último, divisor de enteros (DIV, IDIV). Se añaden algunas instrucciones más como se detalla en la siguiente tabla con sus modos de direccionamiento:

Instrucción	**Modo de direccionamiento**
MOV	Inmediato Directo a registro Directo a memoria
JMP, CALL	Inmediato Directo a registro
ADD, AND, CMPEQ, CMPGE, CMPGT, NOT, OR, SUB, XOR, DEC, DIV, FADD, FMUL, FSUB, IDIV, IMUL, INC, MUL, ROL, ROR, SHL, SHR	Directo a registro Directo a memoria
JZ, JNZ	Relativo a registro (PC)
POP, PUSH	Directo a registro
RET	Inmediato

Se puede añadir más instrucciones que lo aquí visto, pero se deja así debido a que el diseño de la CPU no se especifican el uso de periféricos de entrada/salida, trabajar con cadenas de texto, interrupción a rutinas de servicio, o incluso se podrían añadir otras instrucciones como XCHG que puede ser reemplazada por PUSH, MOV y POP a pesar de que esto último puede necesitar más números de ciclos del reloj.

Diseño del formato de instrucción

El modelo de ejecución es registro-memoria (hay un banco de 16 registros de propósito general y las operaciones de proceso admiten un operando ubicado en memoria)

Mejora: que no admita un operando ubicado en memoria en las operaciones de proceso aprovechando que hay 16 registros de propósito general y su modelo de ejecución será registro-registro, lo que será mucho más rápido ya que no se realizan accesos a memoria (a excepción de MOV o seguir la arquitectura load-store). Aunque se debe de tener en cuenta que la codificación de bits puede ser excesiva.

Los modos de direccionamiento son: inmediato, directo a registro y a memoria, y relativo a registro.

Se pide minimizar el tamaño de representación, es decir, el tamaño del ejecutable debe ser lo menor posible y para ello se usa como posible solución el campo de extensión y, teniendo en cuenta la frecuencia de uso de operaciones mostrado en el libro de Anasagasti, se va a diseñar el formato de instrucción con menos bytes para las operaciones de mayor uso y con más bytes a las operaciones de menor uso. Las operaciones ordenadas de mayor a menor uso y especificando la frecuencia de uso dinámico sobre software comercial son las siguientes:

- Transferencia 42,9%
- Bifurcación 29,6%
- Comparaciones, desplazamientos y lógicas 15,3%
- Aritméticas en binario 7,4%
- Aritméticas en coma flotante 2,6%
- Otras 2,2%

Número de direcciones: Ya que disponemos de 16 registros de propósito general se utilizan instrucciones con múltiples direcciones, intentando que las operaciones se realicen en registros siendo estos más rápidos que en memoria y acelerar la ejecución de los programas. Por lo tanto, se

emplea una combinación de 1 y 2 direcciones. Sin embargo, teniendo en cuenta de que no disponemos de registros de estado, se establecen 3 direcciones para las instrucciones de comparación provocando que posiblemente crezca algo más la longitud de los programas como ocurre con RISC.

Instrucciones de 3 direcciones:

CMPEQ, CMPGE, CMPGT

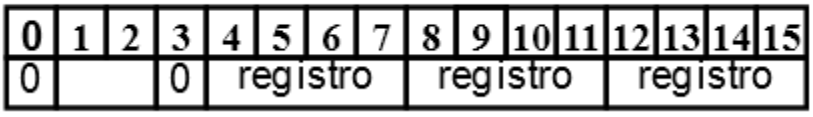
Instrucciones de 2 direcciones:

JMP, CALL, MOV, ADD, AND, OR, SUB, XOR, DIV, FADD, FMUL, FSUB, IDIV, IMUL, MUL, ROL, ROR, SHL, SHR

Instrucciones de 1 dirección:

NOT, DEC, INC, JZ, JNZ, POP, PUSH, RET

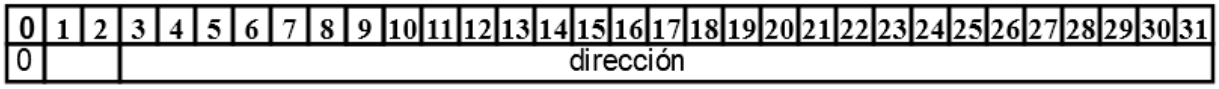
Primero se ha pensado en fijar el tamaño del formato en 16 bits intentando reducir más el tamaño de representación, pero debido a que las instrucciones de comparación requieren de 3 direcciones ya sean, por ejemplo, de tres registros y de que estos mismos necesitarían de 4 bits cada uno, se encuentra como única opción usar el bit 0 para el primer campo de extensión (0), los bits 1 y 2 para las instrucciones CMPcc, el bit 3 para codificar los dos modos de direccionamiento y, por último, dejando el resto para los registros como se muestra a continuación:



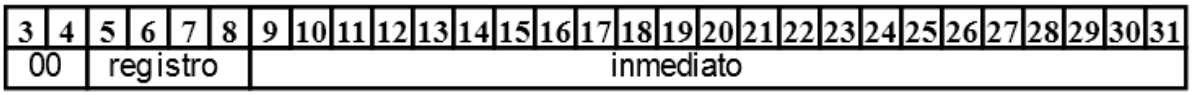
Con todo lo anteriormente visto y como se quiere usar el campo de extensión 0 para las instrucciones de tipo transferencia, se fija el tamaño del formato de 32 bits respectando también así el tamaño de palabra de la CPU.

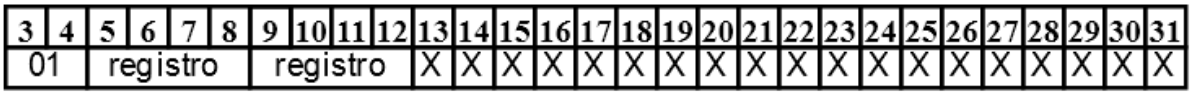
Formato para las instrucciones de tipo transferencia: MOV, PUSH y POP

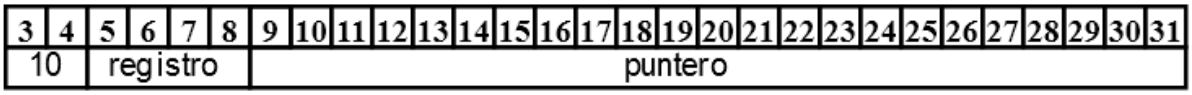
Con el bit 0 como campo de extensión (0), mientras que el bit 1 y bit 2 como opcode. MOV es 00, PUSH 01 y POP 10. (El 11 podría servir para XCHG propuesto anteriormente)

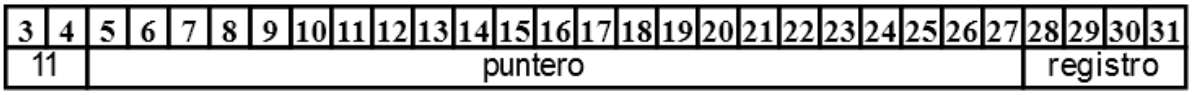


MOV con opcode 00 y sus modos de direccionamiento:

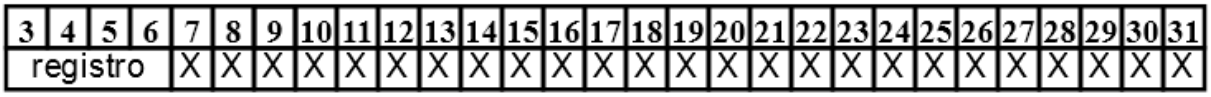






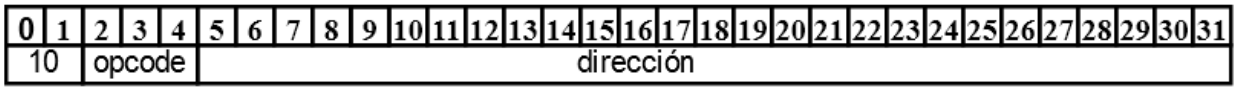


PUSH con opcode 01 y POP con 10:



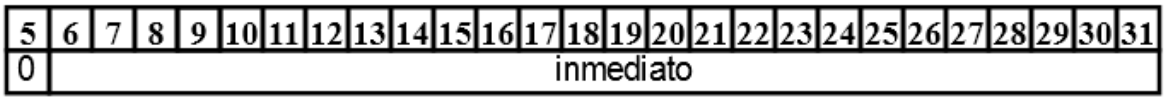
Formato para las instrucciones de tipo bifurcación: JMP, CALL, JZ, JNZ, RET

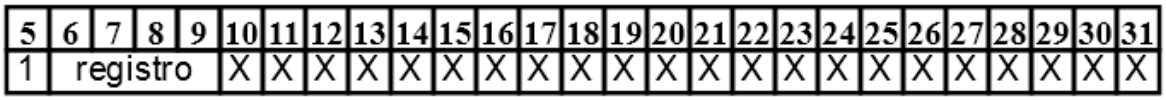
Con el campo de extensión 10.



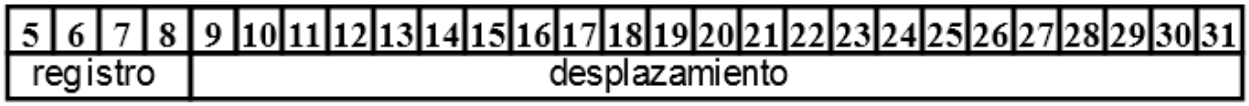
:

JMP con opcode 000 y CALL 001:

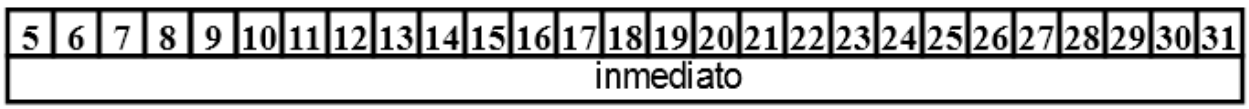




JZ con opcode 010 y JNZ 011:

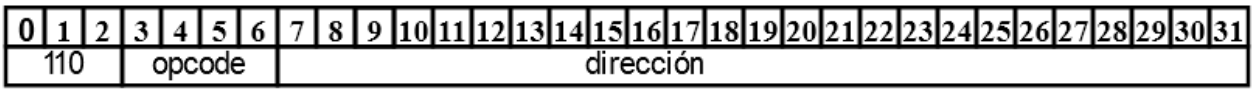


RET con opcode 100:

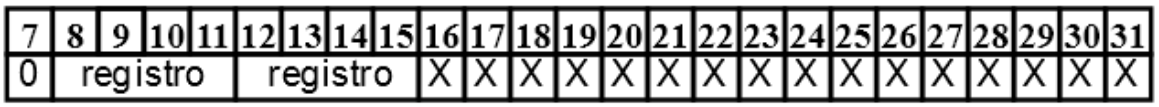


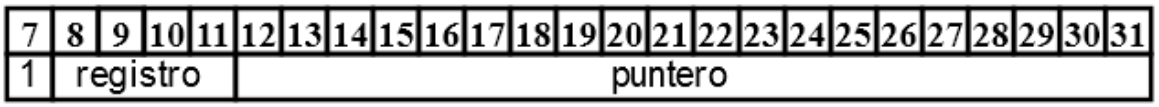
Formato para las instrucciones de tipo comparación, desplazamiento y lógicas: AND, CMPEQ, CMPGE, CMPGT, NOT, OR, XOR, ROL, ROR, SHL, SHR

Con el campo de extensión 110.

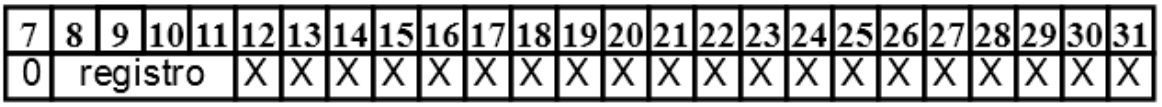


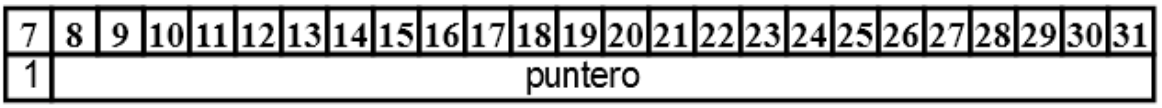
AND con opcode 0000, OR 0001, XOR 0010, ROL 0011, ROR 0100, SHL 0101, SHR 0110:



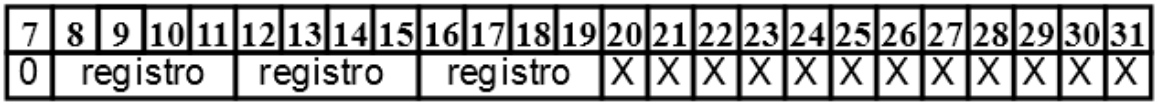


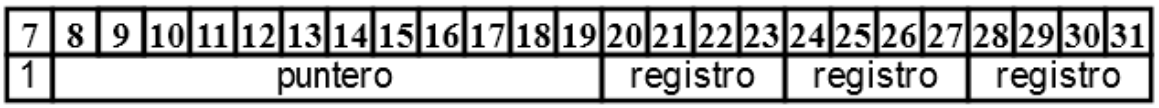
NOT con opcode 0111:





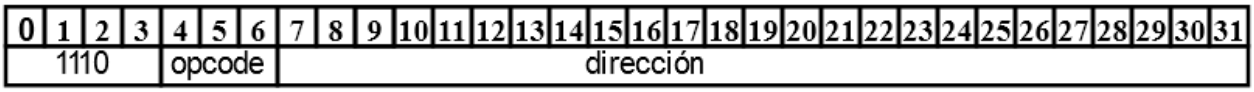
CMPcc con opcode 10cc:



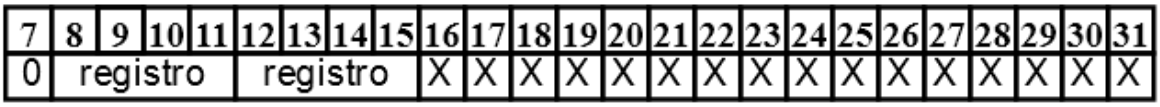


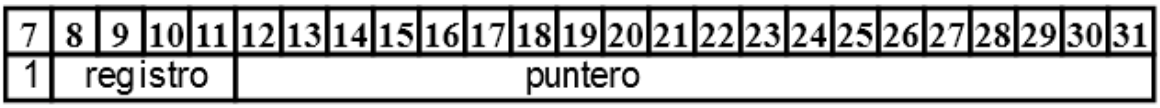
Formato para las instrucciones de tipo aritmética en binario: ADD, SUB, DEC, DIV, IDIV, IMUL, INC, MUL

Con el campo de extensión 1110.

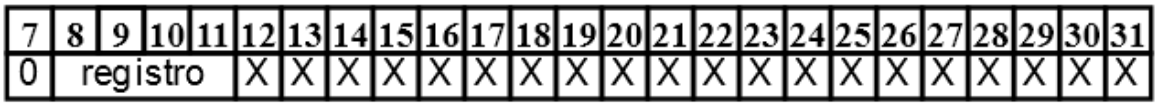


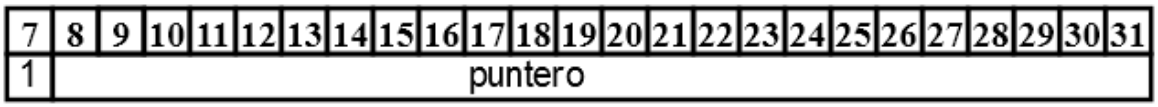
ADD con opcode 000, SUB 001, DIV 010, IDIV 011, MUL, 100, IMUL 101:





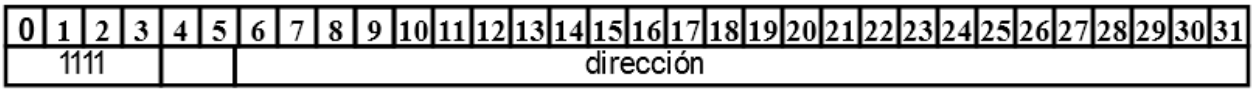
INC con opcode 110 y DEC con 111:



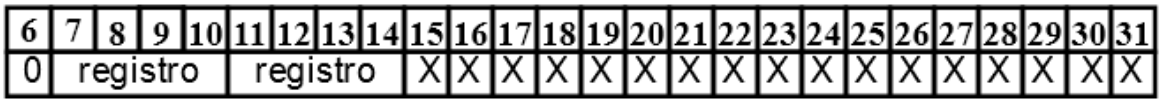


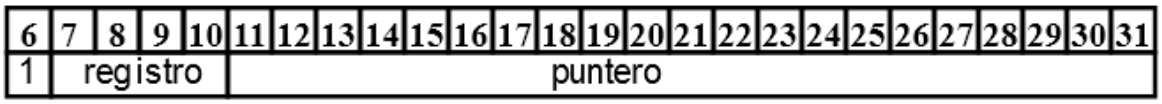
Formato para las instrucciones de tipo aritmética en coma flotante: FADD, FMUL, FSUB

Con el campo de extensión 1111.



FADD con opcode 00, FMUL 01 y FSUB 10:





Como se puede apreciar hay bits sin usar sobre todo en los casos en los que hay operandos de dos registros. Como se ha explicado anteriormente, esto es debido a las instrucciones de comparación que tienen 3 direcciones, por lo que se propone varias mejoras distintas entre ellas:

1º.- Que exista un registro de estado, además aprovechando que la configuración de esta CPU tiene detector de cero, signo y paridad. Esto permitirá que no haya instrucciones de comparación de 3 direcciones y se puedan, seguramente sin ningún problema, tener un formato de instrucción de 16 bits, siendo así el tamaño de representación mucho menor como se propone.

2º.- Que si no hay registro de estado entonces que tenga un modelo de ejecución registro-registro y obviando que las operaciones de proceso admitan un operando en memoria, con un enfoque mayor al cálculo, manteniendo 3 direcciones, con regularidad en el repertorio de instrucciones y en formato de 32 bits como lo visto antes. Lo más cercano a RISC y aprovechando también de las instrucciones en coma flotante como la suma-resta y la multiplicación.

Cálculo del tamaño promedio del repertorio de instrucciones: Con un tamaño fijo de 4 bytes y con 30 instrucciones se obtiene un tamaño de 30 * 4 = 120 bytes

Secuencia de código ejemplo para las siguientes instrucciones:

- Intercambio de dos operandos en memoria:

  MOV R1, [M1] ; R1 = [M1]

  MOV R2, [M2] ; R2 = [M2]

  MOV [M1], R2 ; [M1] = R2

  MOV [M2], R1 ; [M2] = R1

- Suma de dos operandos ubicados en memoria:

  MOV R1, [M1] ; R1 = [M1]

  ADD R1, [M2] ; R1 = R1 + [M2]

  NOTA: En el caso de querer realizar la suma en coma flotante se usaría la instrucción FADD. Como se explica en el libro de Hennessy-Patterson, también se podría tomar como mejora el añadir registros para coma flotante. Esto implica que el número de instrucciones aumente ligeramente, ya que debemos de disponer de un MOV de coma flotante para la transferencia de datos. Sus beneficios son tener el doble de registros sin usar más bits necesarios para los formatos de instrucción, disponer de mayor ancho de banda de registros y poder configurar registros para coma flotante.

- Producto de dos operandos ubicados en memoria:

  MOV R1, [M1] ; R1 = [M1]

  MUL R1, [M2] ; R1 = R1 * [M2]

  NOTA: La multiplicación de dos operandos de 32 bits dará como resultado uno de 64 bits que no puede ocupar en los registros de propósito general al ser de 32 bits (en este caso R1), por lo tanto, se propone como mejora tener dos registros de propósito especial formado por HI y LO para almacenar el resultado: en el registro HI los 32 bits más significados y los otros 32 bits menos significados en el segundo registro especial LO.

- Comparación de dos operandos enteros ubicados en memoria y salto si el primero es mayor que el segundo:

  MOV R1, [M1] ; R1 = [M1]

  MOV R2, [M2] ; R2 = [M2]

  CMPGT R3, R1, R2 ; si R1 > R2 entonces R3 = 1, sino R3 = 0

  JNZ R3, etiquetaSalto ; Salta a etiquetaSalto si R3 no es igual a 0

  NOTA: Debido a que no hay registros de estado se ha decido implementarlo en una combinación entre ALPHA y MIPS. Como hemos visto anteriormente, se necesitan de tres direcciones para que uno de los registros de propósito general haga de flag para luego realizar el salto con JZ/JNZ. Además, como comparar si el primero es mayor que el segundo es lo mismo que comparar si es menor el segundo que el primero se ha decidido que solo es necesario las instrucciones de *mayor que / mayor que o igual* con el objetivo de minimizar el tamaño del repertorio de instrucciones ya que así hay menos instrucciones con 3 direcciones.

Secuencia de invocación de subrutina con especificación del estado de pila en cada paso.

Supongamos que le pasamos como argumentos la base y la altura a la subrutina que tiene que calcular el área del rectángulo, la dirección de memoria pasada como referencia para que la subrutina nos devuelva el área y posteriormente llamamos a la subrutina. Todo ello como en el siguiente ejemplo:

```
MOV R1, B ; R1 = B siendo B cualquier valor numérico que corresponde a la base

MOV R2, H ; R2 = H siendo H cualquier valor numérico que corresponde a la altura

MOV R3, M ; R3 = M siendo M algún valor que sirva como dirección de memoria reservada

PUSH R3 ; Se inserta el puntero a la pila

PUSH R1 ; Se inserta el primer argumento a la pila

PUSH R2 ; Se inserta el segundo argumento a la pila

CALL etiquetaSubrutina ; Transfiere el control a la subrutina
```

Como la pila crece de posiciones altas a bajas, cada vez que se hace un PUSH el puntero de pila (SP) decrementa y lo hace de 4 en 4 debido a que el tamaño de palabra es de 32 bits

La instrucción CALL lo que debe de hacer es insertar en la pila el valor del puntero de instrucciones (IP), también llamado como contador del programa (CP) que servirá como dirección de retorno, insertar el valor del puntero de marco (BP) a la pila, actualizar BP y reservar espacio para variables locales. Esto es equivalente al siguiente código:

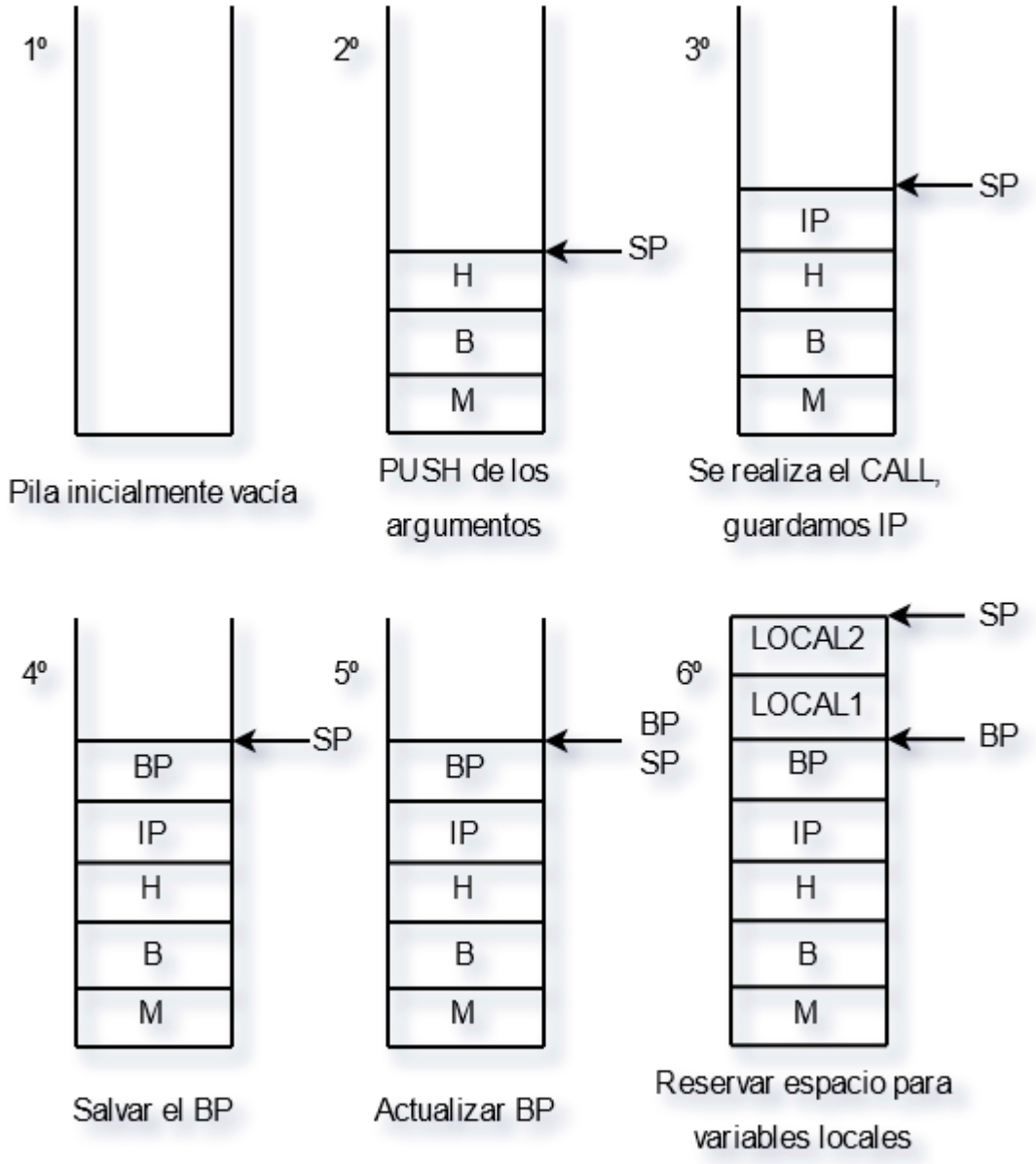
PUSH IP ; Se inserta IP a la pila

PUSH BP ; Se inserta BP a la pila

MOV BP, SP ; BP = SP, actualiza BP

SUB SP, [espacio_para_variables_locales] ; Reserva espacio para variables locales

Con estos datos se ilustra con la siguiente figura el estado de la pila en cada paso mostrando que el puntero de pila (SP) siempre apunta a la posición ocupada tal como se pide:



Secuencia de retorno de subrutina con especificación del estado de la pila en cada paso.

En el retorno lo que va a suceder primero es que se solapan las variables locales, se recupera BP e IP y el *clean-up* lo debe de realizar el llamado, es decir, la subrutina debe de usar la instrucción RET. Este código sirve de ejemplo:

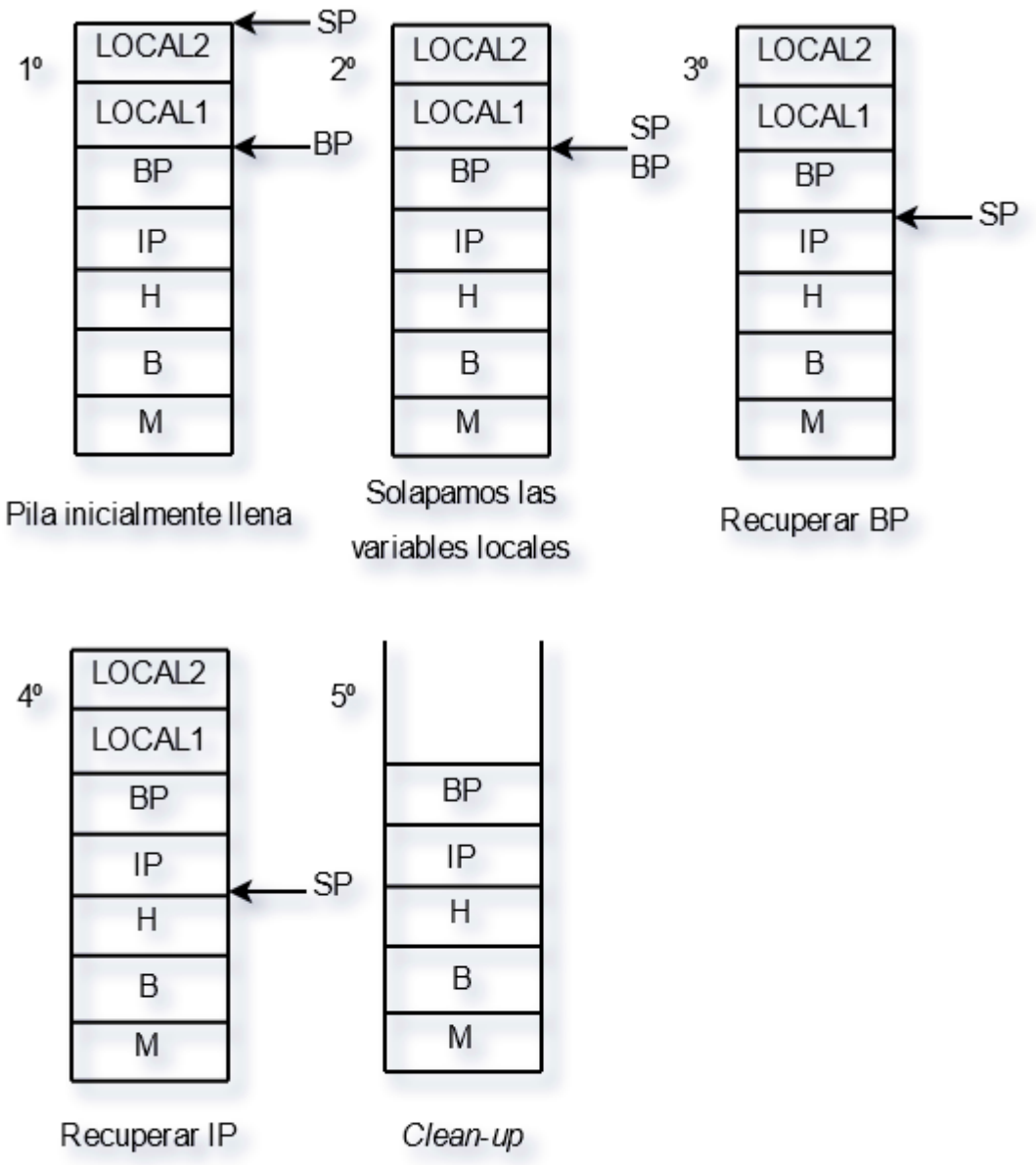
```
MOV SP, BP ; SP = BP, solapamos las variables locales

POP BP ; Extraemos el elemento que apunta SP, eso significa que recuperamos BP y para ello incrementamos SP + 4

POP IP ; Igual que antes pero ahora recuperamos IP

RET 3 ; El llamado usa esta instrucción con un operando inmediato que es el número de argumentos pasados, en este caso, los 3 argumentos que hemos pasado previamente (dirección de memoria, base y altura)
```

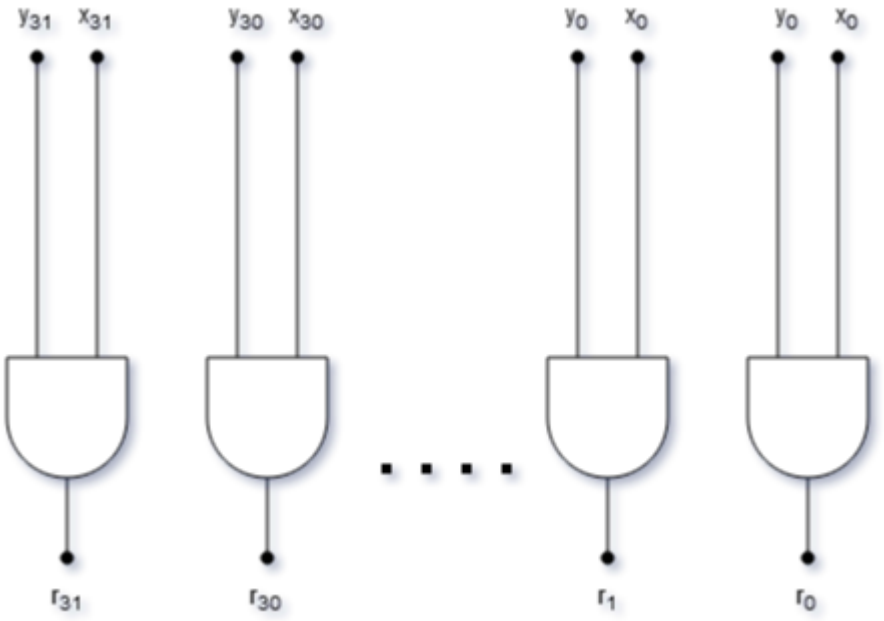
Con la siguiente figura se vuelve a mostrar el estado de la pila en cada paso, pero ahora para el retorno:



Debido a que disponemos de 16 registros de propósito general se propone como mejora la utilización de ventanas de registros y, en este caso, el uso de registros para pasar los argumentos de entrada y salida. Esto es posible ya que, según el libro de Stallings, el número de argumentos que típicamente se usan son 3 hasta un máximo de 6 argumentos, por lo tanto, se podría aprovechar mejor los 16 registros que dispone esta CPU para evitar accesos a memorias y utilizar los registros que son el almacenamiento más rápido que disponemos.

Esquema del operador lógico AND para el tamaño de palabra de 32 bits.

Para ello se colocan las puertas AND en paralelo como se muestra en la siguiente figura:

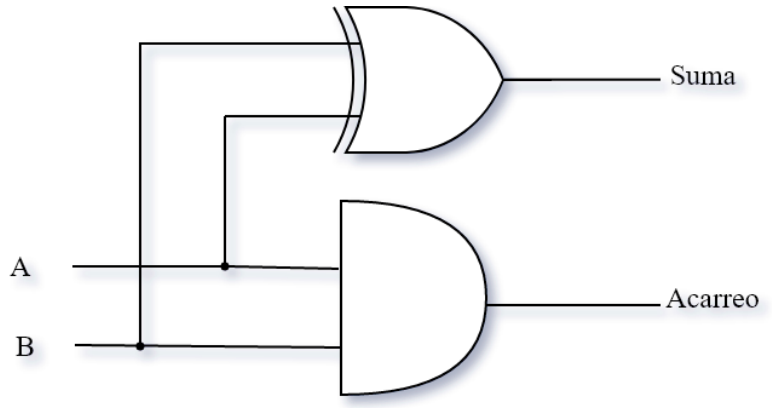


Tiempo de retardo: $32 * r_g$ siendo r_g el retardo genérico de una puerta lógica. Para los otros operadores lógicos de la opción 1 sería lo mismo, pero con sus respectivas puertas lógicas (OR, XOR y NOT)

Esquema del operador aritmético de la suma de 32 bits.

Primero se realiza el esquema del semisumador (*Half-Adder*) siguiendo la siguiente tabla de verdad:

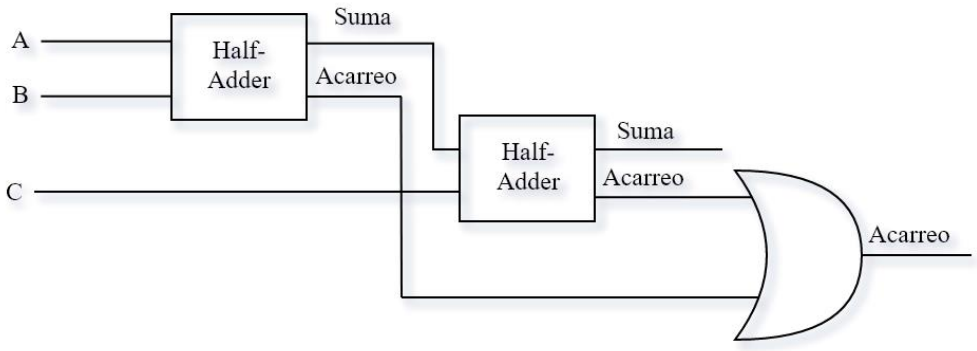
A	B	Suma	Acarreo
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Con tiempo de retardo para la suma $1 * r_g$ y acarreo $1 * r_g$

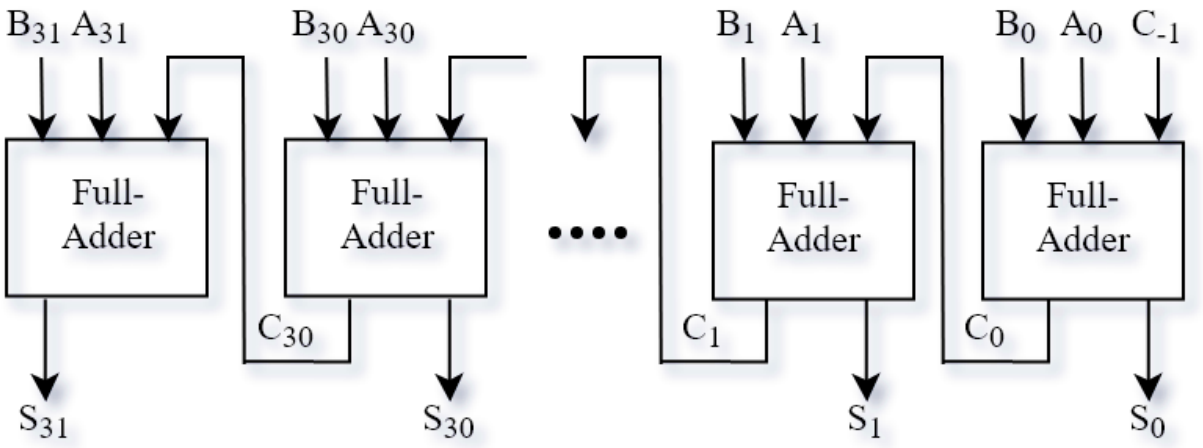
Después se realiza el esquema del sumador completo (*Full-Adder*) partiendo de la siguiente tabla de verdad:

A	B	C	Suma	Acarreo
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Con tiempo de retardo para la suma de $2 * r_g$ y para el acarreo $3 * r_g$

Para la suma en 32 bits:



Tiempo de retardo para la suma de $64 * r_g$ y para el acarreo $65* r_g$ siendo el de peor caso el del acarreo.

Propuesta de mejorar el rendimiento en el diseño del operador suma:

Ya que como se puede apreciar hay un tiempo de retardo provocado por la propagación de los acarreos, se propone como mejora un sumador más utilizado: el de anticipación de acarreo. Esto será con un circuito específico que haga la generación de los acarreos directamente, sin necesidad de tener que propagarse entre todos los sumadores como se ha visto en la figura anterior.

Máxima velocidad de reloj que podría alcanzar la máquina diseñada

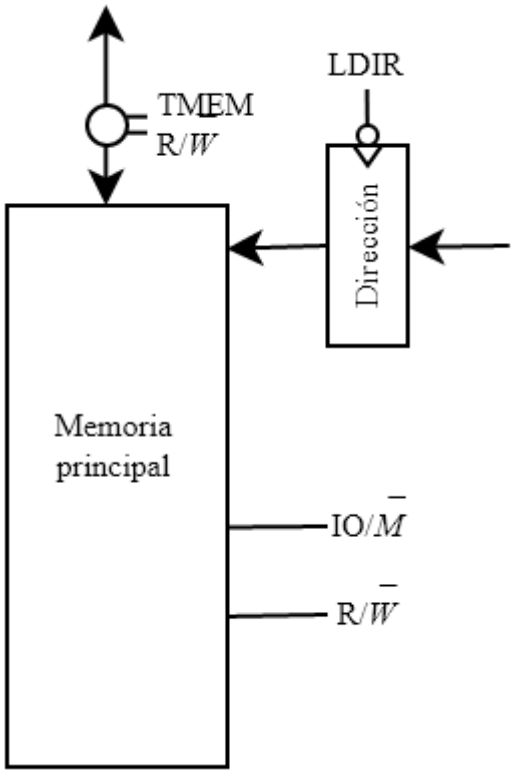
Para saber la máxima velocidad de reloj se tiene que obtener el tiempo de retardo del operador más lento, siendo seguramente que este entre la multiplicación o la división, pero debido a que la multiplicación en coma flotante siguiendo el estándar *IEEE 754* ha resultado algo complejo de obtener, se ha recurrido a estimaciones dadas por el libro *Computer Arithmetic: Volume III*, que establece una media de tiempo de retardo de 5 ns para la multiplicación en coma flotante. Por lo tanto, como valor estimado de la máxima velocidad de reloj que se podría, aproximadamente, alcanzar es de:

$$1/5 = 200 \text{ MHz}$$

Esquema de la máquina propuesta incluyendo ruta de datos, unidad de control, unidades de direccionamiento, banco de registros y bloque de memoria.

Primero se muestran los esquemas de cada parte con sus señales de control para tener un mejor entendimiento y, luego, en su conjunto. Se mantienen los nombres de señales de la propuesta en la hoja de problemas del tema Unidad de Control y así reutilizar la plantilla de cronogramas para el siguiente apartado.

Esquema de la memoria principal



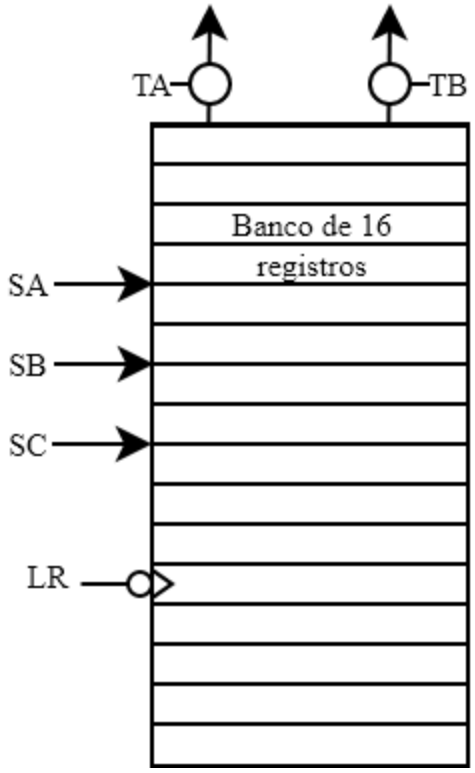
IO/M Inicia ciclo de memoria

R/W Ciclo de lectura/escritura

LDIR Carga una dirección

TMEM Transfiere bidireccionalmente entre la memoria y el bus de datos

Esquema del banco de 16 registros

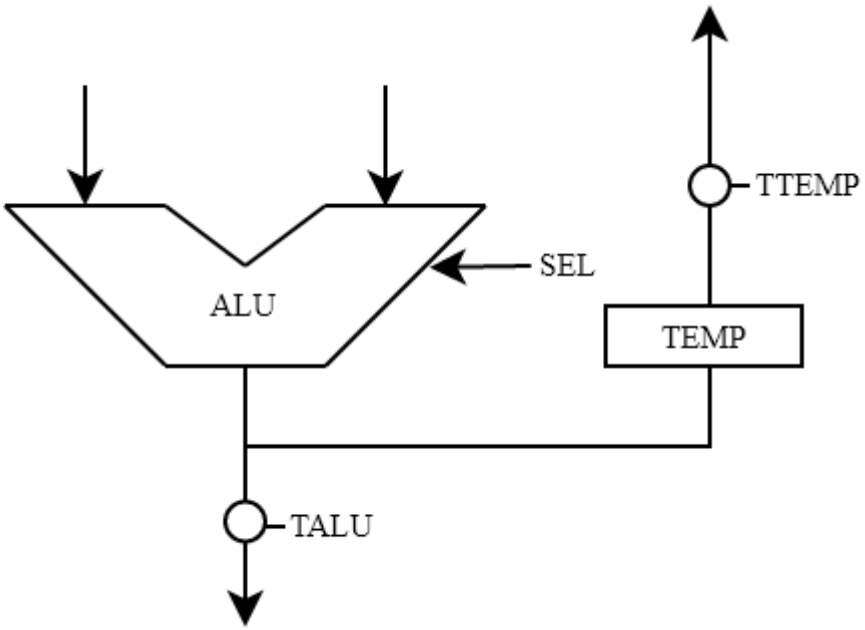


LR Carga el registro que se activa por flanco de bajada

Sx Selecciona un registro

Tx Transfiere el registro

Esquema de la unidad aritmética



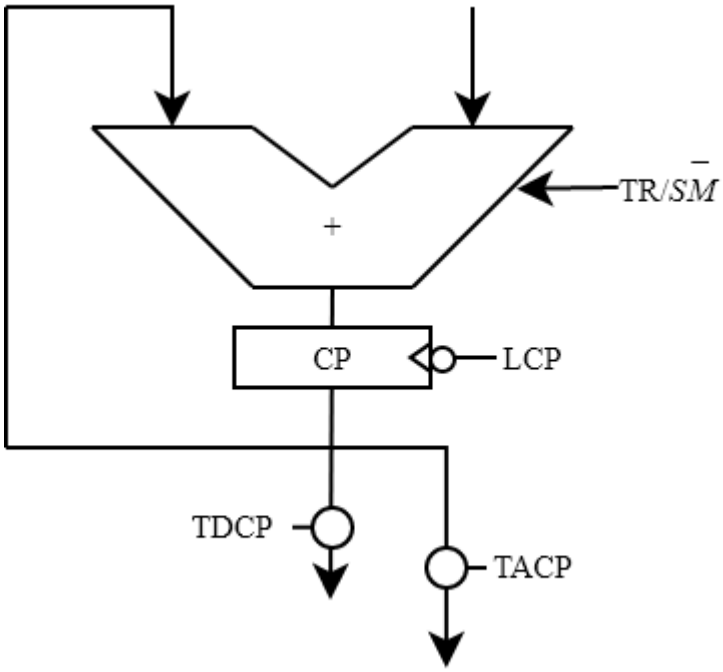
NOTA: Se ha retirado el registro de estado de la ALU como se especifica en la configuración de la CPU, pero aparece la señal LF en el cronograma que no tiene uso alguno.

SEL Selecciona la operación

TALU Transfiere el resultado de la ALU

TTEMP Transfiere el contenido del registro TEMP (para resultados intermedios como la división y multiplicación)

Esquemas de las unidades de direccionamiento

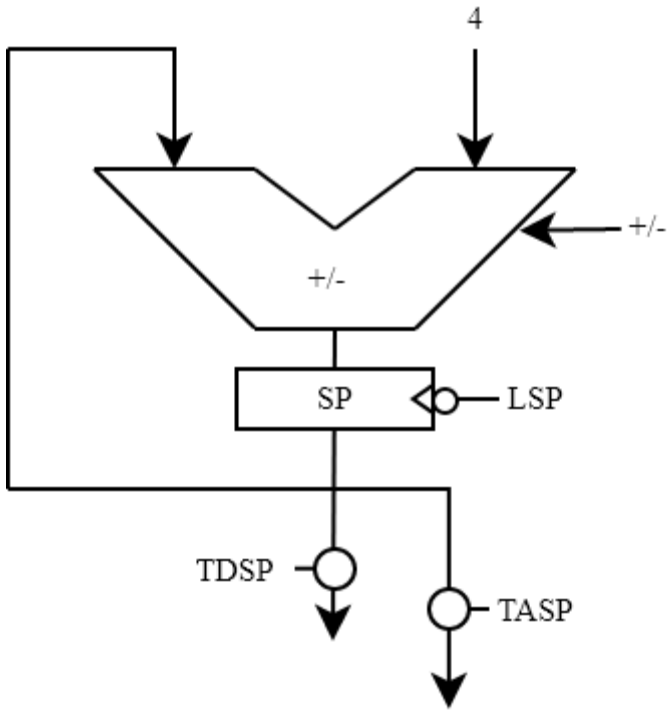


TR/SM Inicia el ciclo de Transferencia/Suma

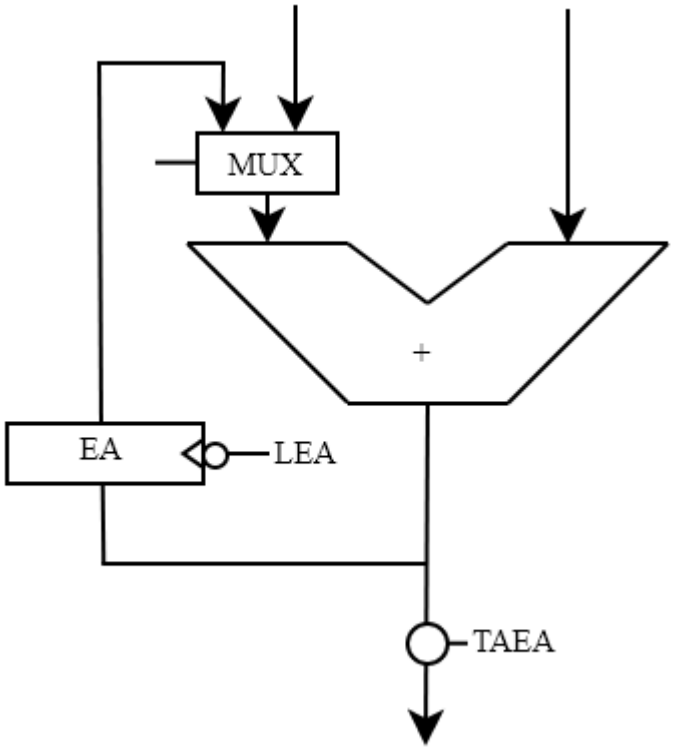
LCP Carga el registro CP (contador del programa) que se activa por flanco de bajada

TDCP Transfiere el registro CP al bus de datos

TACP Transfiere el registro CP al bus de direcciones



+/-	Inicia el ciclo de Suma/Resta
LSP	Carga el registro SP (puntero de pila) que se activa por flanco de bajada
TDSP	Transfiere el registro SP al bus de datos
TASP	Transfiere el registro SP al bus de direcciones

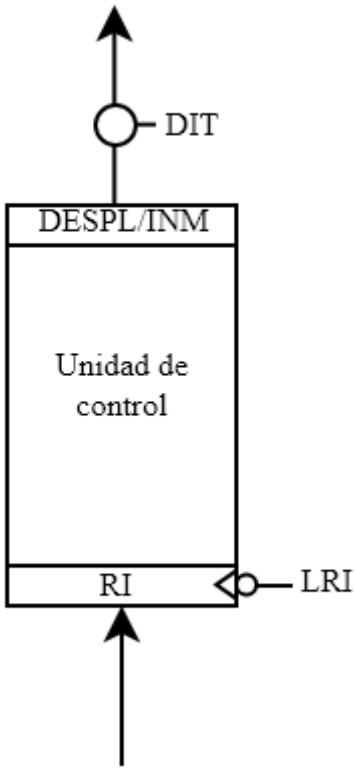


MUX Inicia el multiplexor

LEA Carga el registro EA (*effective address*) que se activa por flanco de bajada

TAEA Transfiere el registro EA al bus de direcciones

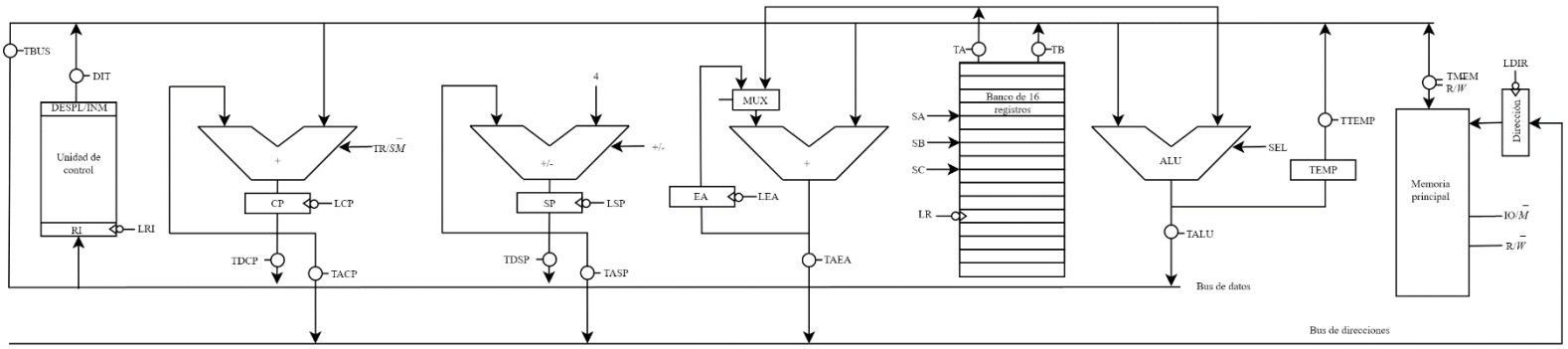
Esquema de la unidad de control



LRI Carga el registro I (instrucción) que se activa por flanco de bajada

DIT Transfiere un dato inmediato o un desplazamiento

Esquema de la ruta de datos



TBUS Transfiere al bus de datos

Cronograma de ejecución de una instrucción de transferencia, de una de proceso y de una de bifurcación.

Instrucción de transferencia: MOV R2, R1

A continuación, se detallan los pasos y después se muestra el cronograma de ejecución.

Buscar la instrucción en la memoria principal:

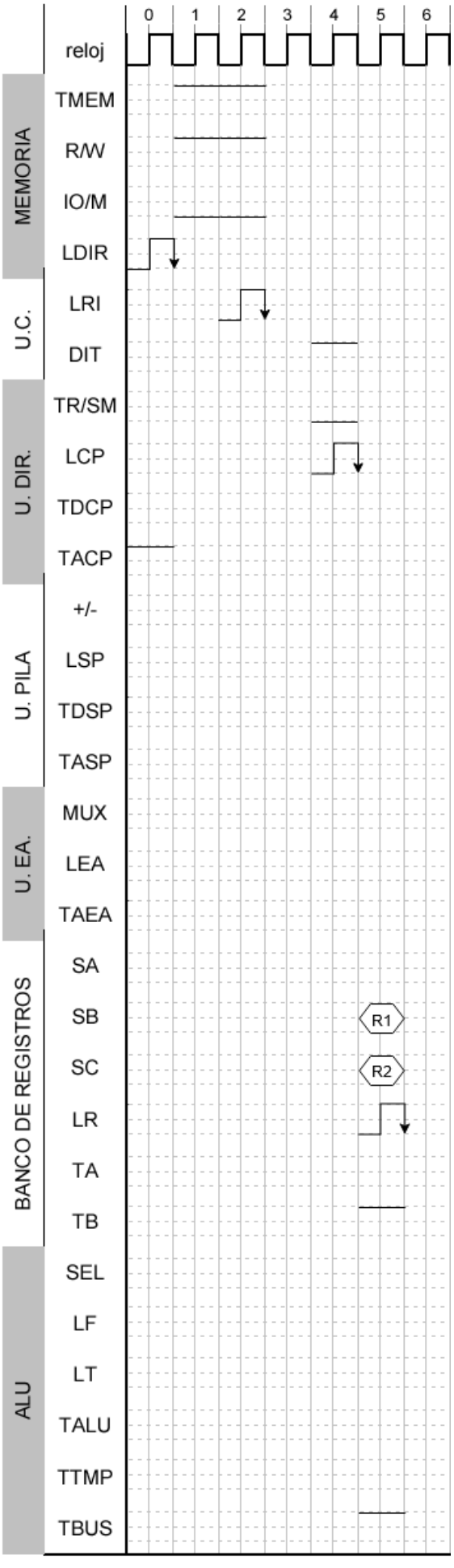
- **TACP:** se activa para pasar CP al bus de direcciones
- **LDIR:** se carga CP al registro de dirección
- **IO/M**, **R/W y TMEM:** se lee la instrucción en memoria
- **LRI:** se carga la instrucción al registro RI

Decodificar la instrucción:

- La unidad de control decodifica la instrucción en un retardo
- **DIT**, **TR/SM y LCP:** se incrementa CP + 4 para ejecutar la siguiente instrucción

Ejecutar la instrucción:

- **LR, SA, SB, TB y TBUS: s**e carga el registro R1 a R2 en un mismo ciclo


0
1
2
3
4
5
6
reloj
MEMORIA
TMEM
R/W
IO/M
LDIR
U.C.
LRI
DIT
U. DIR.
TR/SM
LCP
TDCP
TACP
U. PILA
+/-
LSP
TDSP
TASP
U. EA.
MUX
LEA
TAEA
BANCO DE REGISTROS
SA
SB
SC
LR
TA
TB
R1
R2
ALU
SEL
LF
LT
TALU
TTMP
TBUS

Instrucción de proceso: ADD R2, R1

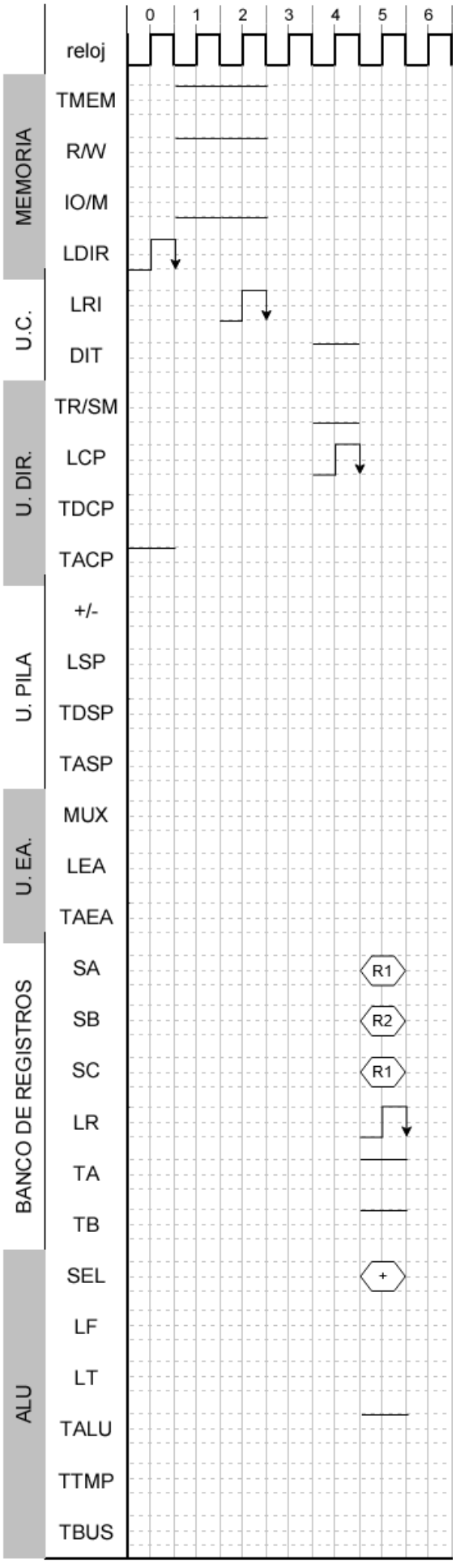
Buscar la instrucción en la memoria principal y decodificar la instrucción siguen los mismos pasos descriptos anteriormente.

Ejecutar la instrucción:

- **LR, SA, SB, TA, TB y SEL:** se realiza la suma de R2 + R1

Guardar resultados:

- **TALU y SC:** se guarda el resultado en R2

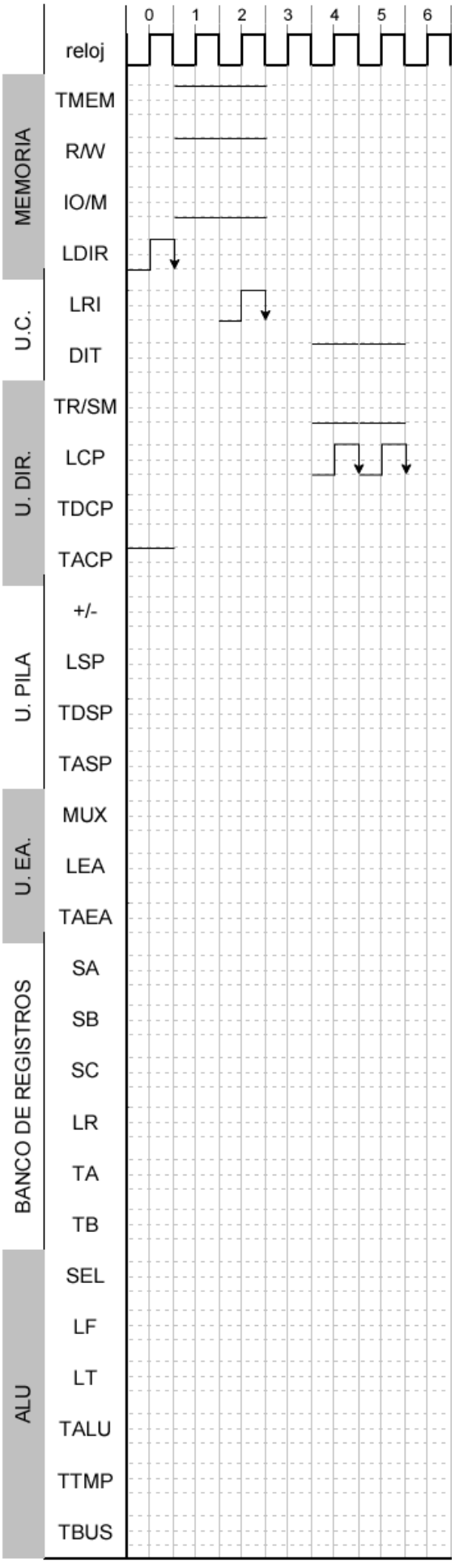

0
1
2
3
4
5
6
reloj
MEMORIA
TMEM
R/W
IO/M
LDIR
U.C.
LRI
DIT
U. DIR.
TR/SM
LCP
TDCP
TACP
U. PILA
+/-
LSP
TDSP
TASP
U. EA.
MUX
LEA
TAEA
BANCO DE REGISTROS
SA
SB
SC
LR
TA
TB
ALU
SEL
LF
LT
TALU
TTMP
TBUS
R1
R2
R1
+

Instrucción de bifurcación: JMP etiquetaInmediato

Buscar la instrucción en la memoria principal y decodificar la instrucción siguen los mismos pasos descriptos anteriormente.

Ejecutar la instrucción:

- **DIT**, **TR/SM y LCP:** se actualiza CP al valor inmediato para ejecutar la siguiente instrucción


0
1
2
3
4
5
6
reloj
MEMORIA
TMEM
R/W
IO/M
LDIR
U.C.
LRI
DIT
U. DIR.
TR/SM
LCP
TDCP
TACP
U. PILA
+/-
LSP
TDSP
TASP
U. EA.
MUX
LEA
TAEA
BANCO DE REGISTROS
SA
SB
SC
LR
TA
TB
ALU
SEL
LF
LT
TALU
TTMP
TBUS

Estimación del CPI de la máquina propuesta para un uso equitativo de todas las instrucciones del repertorio.

Visto anteriormente que una instrucción de transferencia, proceso y bifurcación tarda 6 ciclos en ejecutarse y que se ha tomado como referencia el uso dinámico sobre software comercial:

- Transferencia 42,9%
- Bifurcación 29,6%
- Proceso 27,5%

Se estima que el CPI es el siguiente:

$$CPI = (6 * 42,9 + 6 * 29,6 + 6 * 27,5) / 100 = 6$$

Cálculo del tamaño de la ROM de control en número de celdas en función de las opciones de microprogramación

Las opciones son: microprogramación horizontal que se usan microinstrucciones no codificadas y secuenciamiento explicito que consiste en incluir en cada microinstrucción la dirección de la microinstrucción siguiente.

Supongamos el microprograma para la suma con la instrucción ADD R2, R1 mostrada en la siguiente tabla:

Señal	0	1	2	3	4	5
IO/M	0	1	1	0	0	0
R/W	0	1	1	0	0	0
LDIR	1	0	0	0	0	0
TMEM	0	1	1	0	0	0
LR	0	0	0	0	0	1
SA	0	0	0	0	0	1
SB	0	0	0	0	0	1
SC	0	0	0	0	0	1
TA	0	0	0	0	0	1
TB	0	0	0	0	0	1
SEL	0	0	0	0	0	1
TALU	0	0	0	0	0	1
TTEMP	0	0	0	0	0	0
TR/SM	0	0	0	0	1	0
LCP	0	0	0	0	1	0
TDCP	0	0	0	0	0	0
TACP	1	0	0	0	0	0
+/-	0	0	0	0	0	0
LSP	0	0	0	0	0	0
TDSP	0	0	0	0	0	0
TASP	0	0	0	0	0	0
MUX	0	0	0	0	0	0
LEA	0	0	0	0	0	0
TAEA	0	0	0	0	0	0
LRI	0	0	1	0	0	0
DIT	0	0	0	0	1	0
TBUS	0	0	0	0	0	0

Si tenemos 30 instrucciones en el repertorio entonces se necesitan 5 bits para el código de operación, con 27 señales de control y un contador de un máximo de 32 periodos que necesitan 5 bits, el tamaño será de: $2^5 \; x \; 27 \; x \; 2^5 = 27648 \; bits = 27 \; Kbits$

Como nos pide secuenciamiento explicito, debemos incluir en cada microinstrucción:

- Un campo con la dirección siguiente
- Y un bit para indicar si es la última

Por lo tanto, necesitamos 5 + 5 + 1 adicionales a los de antes y esto es:

$$2^5 \; x \; 27 \; x \; 2^5 + 2^{11} = 29696 \; bits = 29 \; Kbits$$

Con un tamaño de palabra de 32 bits, obtenemos que el número de celdas de la ROM de control es:

29696 / 32 = 928 celdas

Discusión acerca de los puntos fuertes y débiles de la máquina diseñada, así como propuestas de mejora y aceleraciones estimadas (S) si se llevaran a cabo.

A lo largo de este documento se ha propuesto varias mejoras, y haciendo ver los distintos puntos fuertes y débiles que puede tener la máquina. Aquí se proponen otras más:

Ya que el paralelismo es posible y el enfoque de la máquina es para software comercial, se propone como mejora instrucciones SIMD (*Single Instruction, Multiple Data*) que sirven para mejorar el software multimedia, ya que los algoritmos gráficos 3D es común emplear 32 bits para los tipos de datos básicos.

El no disponer registro de estado supone que el detector de cero, signo y paridad pueda ser prescindible, o, como ya se ha visto anteriormente, sería una mejora considerable el tener un registro de estado tanto para no tener instrucciones de 3 direcciones y minimizar el repertorio de instrucciones y para aprovechar mejor los detectores.

Añadir memoria caché: El tener una memoria rápida entre el procesador y la memoria principal puede suponer una mejora notable en el aceleramiento del procesador. Este tipo de memorias son de menor capacidad que la memoria principal pero mucho más rápido.

Para minimizar el tamaño de representación de las instrucciones, la ALU podría disponer de un registro acumulador, de esta forma en ciertas instrucciones puede tener el acumulador como operador implícito y así usar menos bits para representar dichas instrucciones.

Bibliografía

- Andrew Patterson, David y Leroy Hennessy, John

  Estructura y diseño de computadores

  Editorial Reverte – 2011

- Beltrán de Heredia, Jon

  Lenguaje ensamblador de los 80x86

  Anaya Multimedia – 2003

- de Frutos Redondo, José Antonio y Rico López, Rafael

  Arquitectura de computadores

  Universidad de Alcalá Servicio de Publicaciones – 1995

- de Miguel Anasagasti, Pedro

  Fundamentos de los computadores

  Ediciones Parainfo S.A – 2004

- Eugene Swartzlander, Earl

  Computer Arithmetic: Volume III

  World Scientific Publishing Co., Inc. - 2015

- Stallings, William

  Organización y arquitectura de computadores

  Grupo Anaya Publicaciones Generales – 2006