

Estructura de Computadores

Apuntes colaborativos

Año: 2013-2017

Apuntes colaborativos que resumen la asignatura de Estructura de Computadores.

Se pueden modificar todo sin ningún problema.

Todo cambio a mejor es bienvenido.

NOTA: Tomar los apuntes con cautela, puede que haya cosas mal.

El creador y autores no se hacen responsables de lo que pueda contener este documento ni de cambios realizados por terceros.



[EC ApuntesColaborativos FIWIKI](#) is licensed under a [Creative Commons Reconocimiento-NoComercial-CompartirIgual 4.0 Internacional](#). Creado a partir de los [apuntes de la asignatura](#) y las clases presenciales a lo largo del tiempo.

Licencia

Información sobre la licencia:

- <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Autores

Nombre	Twitter / web / etc..
Diego Fernández	@diegofpb
Roberto Garrido	@Garri23_23
Sergio Valverde	@svg153
Batmafia	
...	...

Reconocimientos

- [DATSI] - <http://www.datsi.fi.upm.es/>.

ÍNDICE

[TEMA 1 - INTRODUCCIÓN A LOS COMPUTADORES](#)

[Computador Von Neumann](#)

[Memoria principal](#)

[Operación de lectura de memoria principal](#)

[Operación de escritura en la memoria principal](#)

[Organización del espacio de memoria](#)

[Se basa en el Principio de Localidad Espacial y Temporal y el de las Leyes Dinámicas. En resumen dicen que es muy probable ejecutar instrucciones cercanas y repetir instrucciones.](#)

[Los parámetros característicos de la memoria principal](#)

[Unidad central de proceso \(CPU\)](#)

[Unidad aritmético-lógica \(ALU\)](#)

[Registros de propósito general](#)

[Registros de propósito específico](#)

[Registros transparentes](#)

[Unidad de control](#)

[La instrucción de carga de un dato en un registro.](#)

[La instrucción de resta.](#)

[La instrucción de suma.](#)

[Unidad de Entrada/Salida](#)

[Direccionamiento de los dispositivos:](#)

[Modos de realizar la operación de transferencia de E/S:](#)

[Software de sistemas](#)

[Compiladores y ensambladores](#)

[Cargadores \(bootstrap\)](#)

[Sistema operativos](#)

[Parámetros característicos](#)

[Ejemplo](#)

[TEMA 2 - PROGRAMACIÓN EN ENSAMBLADOR](#)

[Introducción](#)

[Lenguaje Máquina](#)

[Juego de instrucciones](#)

[Tipos de datos](#)

[Modos de direccionamiento](#)

[Tipo de direccionamiento](#)

[Ejemplo](#)

[Juego de instrucciones](#)

[Transferencia de datos](#)

[Saltos \(BRANCH\) o bifurcaciones](#)

[Aritméticas](#)

[Lógicas](#)

[Desplazamientos](#)

[De bit Lógicas](#)

[Ejercici](#)

[Ejercicios y Problemas \(Instrucciones y direccionamientos\) \(inst_dir_12-13.pdf\)](#)

[La mayoría de los ejercicios de esta hoja de problemas están resueltos en otro doc en la teoría del tema dos llamado igual que el pdf.](#)

[8](#)

[Problema](#)

[Otros](#)

[1.](#)

[Arquitectura 88110](#)

[Procesador](#)

[Memoria principal](#)

[Modos de direccionamiento](#)

[Direccionamiento directo a registro](#)

[Direccionamiento inmediato](#)

[Direccionamiento relativo a registro base](#)

[Direccionamiento relativo a PC](#)

[Direccionamiento indirecto a registro](#)

[Direccionamiento campos de bit](#)

[Juegos de Instrucciones](#)

[Instrucciones lógicas](#)

[Instrucciones aritméticas](#)

[Bifurcaciones/saltos](#)

[Transferencia \(memoria\)](#)

[Campos de bit](#)

[Instrucciones de coma flotante](#)

[Ensamblador/Cargador](#)

[Pseudoinstrucciones](#)

[Macroinstrucciones \("macros"\)](#)

[Ejemplos](#)

[Ejemplo](#)

[Instrucciones parecidas](#)

[Ejemplo "data"](#)

[Ejercicio](#)

[Ejercicios y Problemas \(Instrucciones y direccionamientos\) \(inst_dir_12-13.pdf\)](#)

[11](#)

[Problema](#)

[Ejemplos \(Drive\)](#)

[Ejemplo 1: Vectores](#)

[Ejemplo 2: Matrices](#)

[Ejemplo 3: Lista no ordenada y compacta](#)

[Ejemplo 4: Lista ordenada y compacta](#)

[Subrutinas](#)

[PXXXXX a la subrutina:](#)

[Tipos de variables que utiliza una subrutinas](#)

[Paso de parámetro a la subrutina:](#)

[Donde pasamos los parámetros:](#)

[En el 88110](#)

[Marco de pila:](#)

[Puntero de marco de pila](#)

[Ejemplo](#)

[Factorial en el 88110](#)

[La lista esta almacenada](#)

[TEMA 3 - PROCESADOR](#)

[Objetivos](#)

[Introducción](#)

[Unidad de control](#)

[Reloj](#)

[Problemas](#)

[Hoja de uc problemas13-14_s](#)

[Problema 1](#)

[Problema 3](#)

[Problema 4](#)

[Problema 8](#)

[Problema 15](#)

[TEMA 4 - REPRESENTACIÓN Y ARITMÉTICA](#)

[Representación de la información](#)

[Representaciones alfanuméricas](#)

[Representaciones numéricas](#)

[Limitaciones de una representación](#)

[Cambio de base](#)

[Operaciones lógicas](#)

[Operaciones](#)

[Representación en coma fija](#)

[Binario sin signo](#)

[Complemento a 2](#)

[Complemento a 1](#)

[TEMA 5 - PERIFÉRICOS](#)

[FALTA LO PRIMERO](#)

Página en blanco

TEMA 1 - INTRODUCCIÓN A LOS COMPUTADORES

Computador Von Neumann

- ★ Ejecuta instrucciones de forma secuencial (una tras otra).
- ★ Las instrucciones y los datos están almacenados en la memoria principal de la máquina para que puedan ser ejecutadas.

Una **instrucción** es un conjunto de bits (01010111000... 1 lógico representa el nivel alto de energía de señal y el 0 lógico representa el nivel bajo) directamente interpretables por el ordenador. Contiene la siguiente información:

- Operación a realizar.
- Los datos sobre los que opera la instrucción o su ubicación en memoria principal.
- Lugar en el que se deposita el resultado.

Una instrucción es **autocontenida** porque contiene información sobre dónde están los datos, qué tiene que hacer y dónde deben depositarse los resultados.

Un computador puede interpretar y ejecutar un conjunto limitado de instrucciones. A ese conjunto se le llama **juego de instrucciones**. Cada computador tiene su propio juego de instrucciones y su propio código máquina.

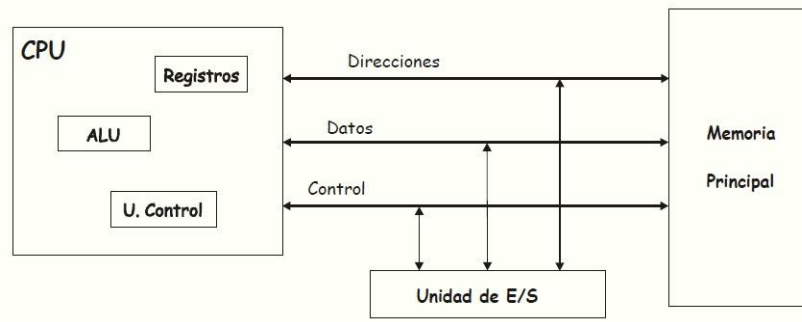
El **computador Von Neumann** es un modelo de computación basado en tres conceptos básicos:

- Los datos y las instrucciones del programa a ejecutar están almacenados en una única memoria de lectura/escritura (R/W), que es la **memoria principal** (MP). Se le llama **arquitectura de programa almacenado**.
- El contenido de la memoria principal es **accesible por direcciones**. Quiere decir que, para acceder a dicha memoria de lectura/escritura, sólo hay que dar la dirección de una posición de memoria.
- La ejecución de las instrucciones se realiza de manera **secuencial**.

El modelo de computación del computador Von Neumann está formado por tres unidades principales: **CPU**, Memoria principal (**MP**) y módulo de entrada/salida (**E/S**).

Las tres unidades principales están relacionadas por un conjunto de buses. Hay buses de tres tipos, principalmente: de direcciones, de datos y de control.

- **Bus de direcciones:** Por donde pasan las direcciones de memoria principal a la que se quiere acceder.
- **Bus de datos:** Sirven para transmitir los datos de lectura, escritura al procesador o a la memoria.
- **Bus de control:** Gestiona todos los elementos de la máquina mediante señales eléctricas.

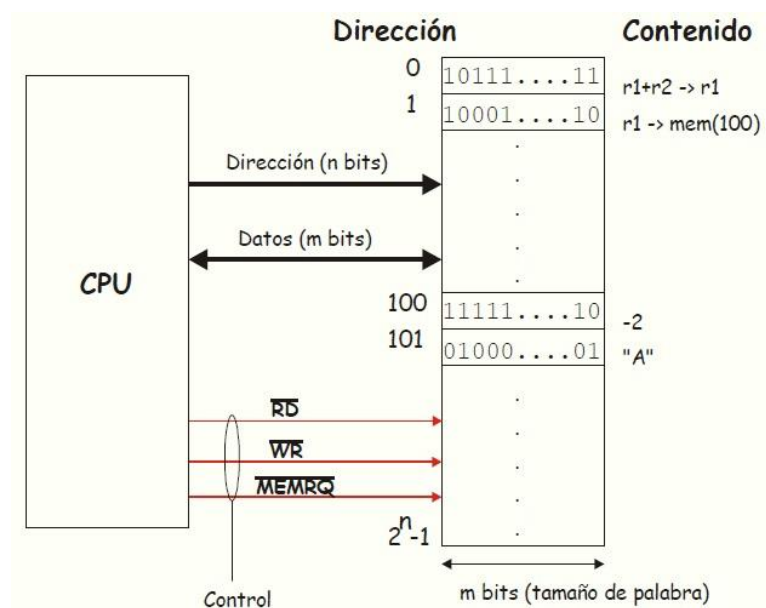


Memoria principal

Está formada por un conjunto de celdas o posiciones de memoria (palabras) con un tamaño determinado (número de bits) para toda la memoria. Almacena datos e instrucciones.

A las palabras de la memoria principal se puede acceder de dos maneras:

- Directamente a palabra, que se llama **direccionamiento a palabra**.
- **Direccionamiento a byte (1 Byte - 8 bits)**.



El procesador lanza la dirección por el bus de direcciones para acceder a memoria con los

bits correspondientes a la dirección. Los datos viajarán por el bus de datos, mientras que por el bus de control se transmitirá la señal de lectura, la de escritura y la petición de memoria.

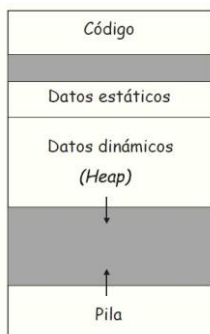
Operación de lectura de memoria principal

El procesador sitúa en el bus de direcciones la dirección de la palabra (dirección de alguna celda de la memoria principal) que se quiere leer y activa la señal (es decir, pone a 1 esa señal) MEMRQ (Petición de acceso a memoria) y RD (señal de lectura). Después de un cierto tiempo llamado **tiempo de acceso a memoria principal**, la memoria vuelca lo que tiene en esa dirección (datos u otra dirección) en el bus de datos y el procesador los coge para su uso.

Operación de escritura en la memoria principal

El procesador sitúa en el bus de direcciones la dirección de la palabra (dirección de alguna celda de la memoria principal) donde se quiere escribir (la escritura es destructiva) y activa la señal (es decir, pone a 1 esa señal) MEMRQ (Petición de acceso a memoria) y WR (señal de escritura). Después de un cierto tiempo llamado **tiempo de acceso a memoria principal** se escribe en esa dirección.

Organización del espacio de memoria



- Instrucciones del programa escrito en código máquina.
- Datos estático. Ej. var. Globales
- Datos dinámicos (Heap). organizado mediante una estructura de datos.
- Pila, que es una zona de memoria que sirve para ir almacenando información para más tarde ir extrayéndola, siguiendo su estructura FIFO. No todos los computadores tienen definida una pila, pero el usuario la puede definir usando los registros

Se basa en el [Principio de Localidad Espacial y Temporal](#) y el de las Leyes Dinámicas. En resumen dicen que es muy probable ejecutar instrucciones cercanas y repetir instrucciones.

Los parámetros característicos de la memoria principal

- **Capacidad:** Se especifica por el número de direcciones ($2^n - 1$) por el número de bits de cada dirección (m). En vez del número de bits se puede hablar del número de bytes.

$$(2^n - 1) \cdot m$$

Los [tamaños típicos](#) de de la memorias se encuentran entre 256MB y 16GB.

- **Tiempo de acceso:** Es el tiempo total desde que el procesador lanza una operación de lectura o escritura hasta que se produce esa lectura o escritura. Se encuentra entre 60 a 100 ns. Varía dependiendo de la tecnología de la memoria.

$$1 \text{ ms} = 10^{-3} \text{ s}$$

$$1 \mu\text{s} = 10^{-6} \text{ s}$$

$$1 \text{ ns} = 10^{-9} \text{ s}$$

$$1 \text{ ps} = 10^{-12} \text{ s}$$

$$1 \text{ fnts} = 1 \text{ fs} = 10^{-15} \text{ s} \text{ (femtosegundo)}$$

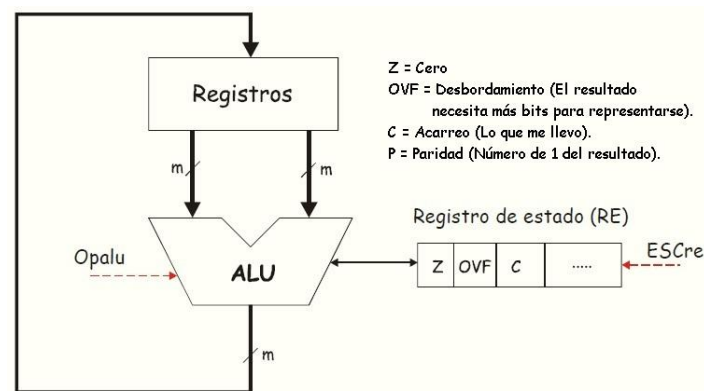
- **Tamaño de la palabra:** Habitualmente se indica con bytes (1 Bytes = 8 bits). los tamaños más típicos son: 4 Bytes = 32 bits y 8 Bytes = 64 bits.

Unidad central de proceso (CPU)

La Unidad Central de Proceso (Central Processing Unit, CPU), está formada por tres componentes: ALU, Unidad de control y Registros.

- **ALU:** Unidad aritmético-lógica. Realiza operaciones aritméticas y lógicas implicadas en las instrucciones.
- **Unidad de control:** Genera las órdenes de control que se envían a todos los elementos del computador para que realicen su función. Lo hace mediante las señales implicadas en el bus de control.
- **Registros:** El procesador contiene un conjunto de registros con características y funciones diferentes, pero que todos son elementos de almacenamiento temporal.

Unidad aritmético-lógica (ALU)



La **unidad aritmético-lógica** (*arithmetic logic unit, ALU*) es un circuito digital donde se realizan todas las **operaciones elementales** de las instrucciones (suma, resta, desplazamiento y operaciones lógicas). Se hace sobre cadenas de bits de longitud fija, generalmente palabras. Eso quiere decir que la longitud de las palabras siempre será la misma, trabajará sobre una palabra de **bits fijos**, no puede variar, ya sea de 16, 32 o 64 bits. Los datos provienen de la memoria o los registros. Éstos han de tener los mismos bits.

La ALU tiene un conjunto de entradas "**Opalu**" que le indica la operación a realizar.

La ALU está directamente relacionada con el **registro de estado (RE)**, ya que almacena la estado de la última operación de las operaciones aritmético lógicas para por ejemplo las instrucciones de salto.

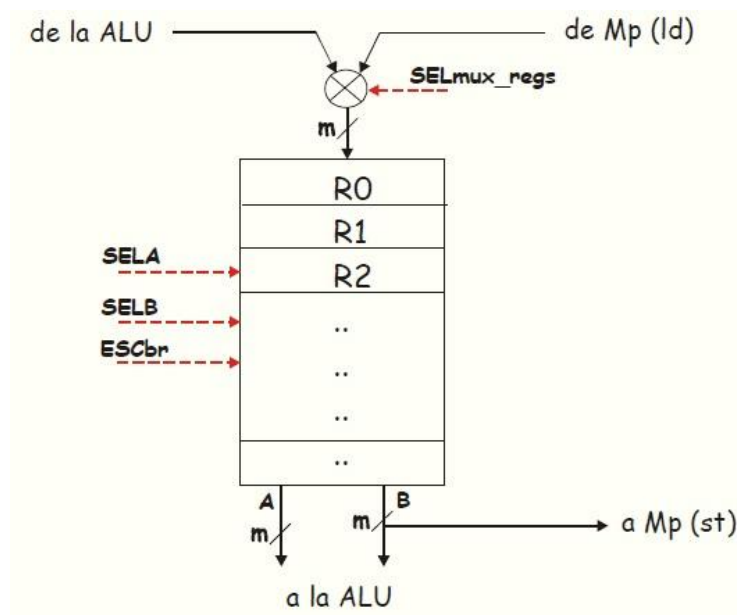
$r2 \leftarrow r2 - r3$; $r5 \leftarrow r5 - r4$; saltar si z es 1 a FIN

Modelo de ejecución del computador: La ALU está relacionada con el modelo de ejecución del computador. El modelo de ejecución del computador viene definido por el lugar donde se encuentran los operandos de una instrucción aritmético-lógica. Hay tres posibles

modelos de ejecución:

- *Registro - Registro*: Actualmente el más habitual es éste. Indica que los operandos lo toman desde el registro. Es el más rápido.
- *Registro - Memoria*: Un operando está en un registro y el otro puede provenir de la memoria. Pero también los dos de registros
- *Memoria - Memoria*: Los dos operandos de las instrucciones están en memoria. Pero también pueden venir tanto del registro como de la memoria.

Como podemos ver se van solapando los modelos. Es decir, el modelo *Memoria - Memoria* comprende también los dos anteriores.



Se pueden clasificar en tres tipos.

- De **propósito general**: Se encuentran agrupados en el banco de registros.
- De **propósito específico**: Los fundamentales son el PC (Program Counter), SR (State Register), SP (Stack Pointer).
- **Transparentes**: IR (instruction register), AR (address register) y DR (data register).

Para seleccionar el registro que va a salir por el puerto a = SELA , puerto b = SELB.

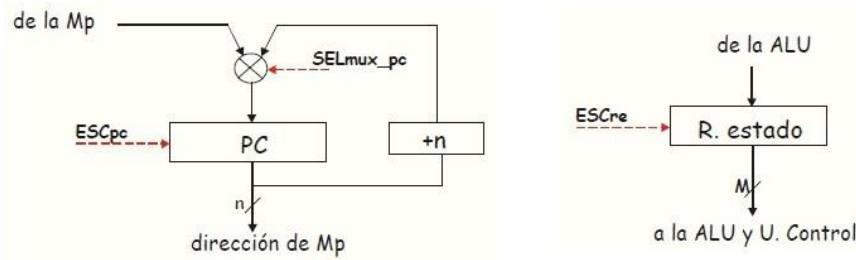
Registros de propósito general

Puede ser utilizado por cualquier instrucción que se lleve a cabo sobre datos, ya que no tienen una utilidad determinada o fija, ya que depende de como los utilice el programador en el programa.

Esos registros que aparecen en las instrucciones son estos registros del banco de registros. Su tamaño es el de la palabra del computador.

Registros de propósito específico

Su uso está restringido a determinadas instrucciones. Se usan en labores concretas en el computador. Suelen utilizarse de manera implícita en las instrucciones. El programador lo conoce pero no puede modificarlos.

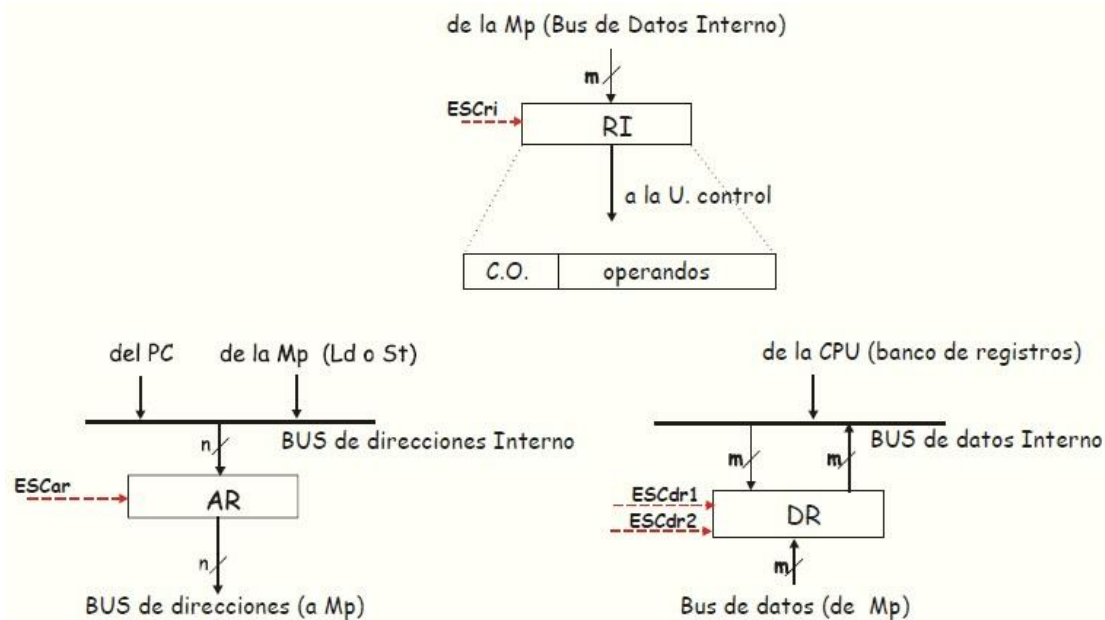


- **Registro contador de programa (Program Counter, PC):** contiene siempre (no siempre, en el 88110, no) la dirección de memoria principal en la que está la siguiente instrucción a ejecutar por el computador. Cuando una instrucción se está ejecutando el contador se incrementa. Si el direccionamiento es a nivel de palabra se incrementa en 1 y si es a byte dependerá del tamaño de la palabra. En las instrucciones de salto y bifurcación, que rompen esa secuencia del programa, se cargará en PC la dirección a la que se salta, es decir, la que se va a ejecutar a continuación.
- **Registro de estado (State Register, SR):** Registro asociado a la ALU de propósito específico compuesto por biestables llamados **flags** que almacena el estado de la última operación realizada sobre la ALU, pero solo de instrucciones aritméticas y lógicas (por ejemplo, los desplazamientos directos relativos a Reg. base no se modifican), ya que algunas operaciones son condicionales y dependen del resultado de la operación anterior, las más típicas son las operaciones de salto que se usa para realizar el secuenciamiento de las instrucciones. Este registro se analiza y tiene sentido bit a bit, contiene información sobre el resultado de la última operación realizada por la ALU. Dependiendo de la máquina, el registro de estado puede tener una longitud u otra.



- **Registro de puntero de pila (Stack Pointer, SP):** El puntero de pila o Stack Pointer contiene siempre una dirección de memoria principal que es en la que está el último elemento almacenado en la pila o la primera posición libre de la pila (la dirección de la cima). En algunas máquinas no existe este registro por qué no está implementada la pila, pero puede implementarse mediante un registro de propósito general. Como es el caso del 88110 de Motorola.

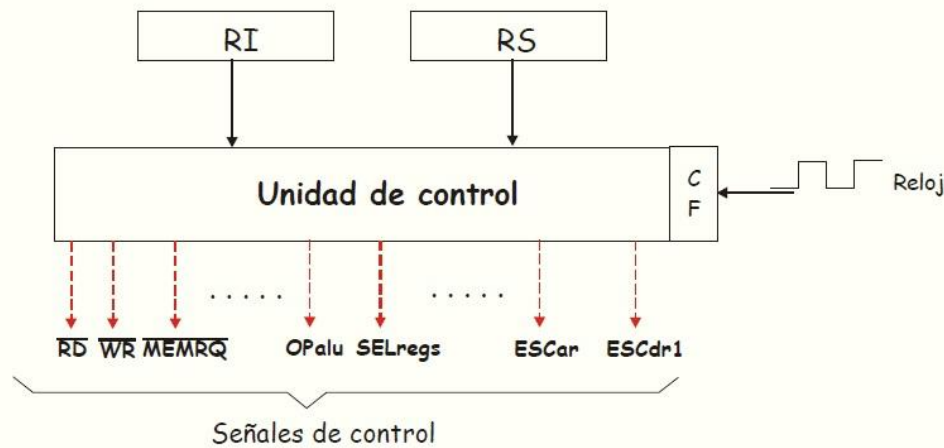
Registros transparentes



Usa internamente el procesador y por lo tanto no se especifican en las instrucciones. Son transparentes al programador, ya que no puede utilizarlos:

- **Registro de instrucción (Instruction Register, IR):** Cuando el procesador va a ejecutar una instrucción (que se encuentra en la memoria principal, por la segunda definición de Von Neumann), lo que hace es acceder a memoria usando la dirección de memoria que se encuentra en el PC (ya que, la siguiente instrucción a ejecutar se encuentra en el PC), la memoria coge la instrucción la vuelca en el bus de datos y se carga en este registro de instrucción que se encuentra en el procesador, que permanece ahí durante todo el tiempo que dura la instrucción. Posteriormente se actualiza el PC, para poder acceder a la siguiente instrucción o a la segunda palabra de la instrucción, dependerá si la instrucción que hemos cargado en el IR ocupa más de una palabra, si es así, la segunda palabra se encontrará en la siguiente posición de memoria (utilizan direcciones de palabras contiguas) en memoria principal.
- **Registro de direcciones (Address Register, AR):** Almacena la dirección de memoria a la que se quiere acceder, ya sea para escribir o para leer de memoria. Esta información se puede cargar desde el procesador si se recibe la señal correcta de la unidad de control para poder almacenar la dirección, aunque también se puede almacenar desde el registro de instrucciones.
- **Registro de datos (Data Register, DR):** Almacena la información que se va a escribir o leer en memoria. Hay dos señales, de carga o de escritura, es decir, una para cuando se va a leer y otra para cuando se va a escribir. Se puede cargar el registro desde el procesador en una operación de escritura o desde la memoria principal en una operación de lectura, con las señales de control correspondientes.

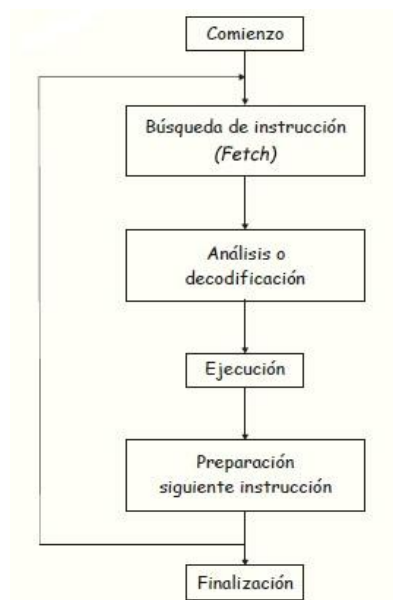
Unidad de control



Es la encargada de ir leyendo las instrucciones de la memoria principal de manera síncrona (en cada ciclo o periodo de reloj), y de controlar todas las señales de control para que las instrucciones se ejecuten en cada instante. De analizarlas, decodificarlas y de darles órdenes a todos los componentes del computador para que se ejecuten.

La unidad de control recibe la información de los registros de instrucciones y el de estado.

¿Qué pasos se siguen para ejecutar una instrucción? Esto son las fases de ejecución de una instrucción.

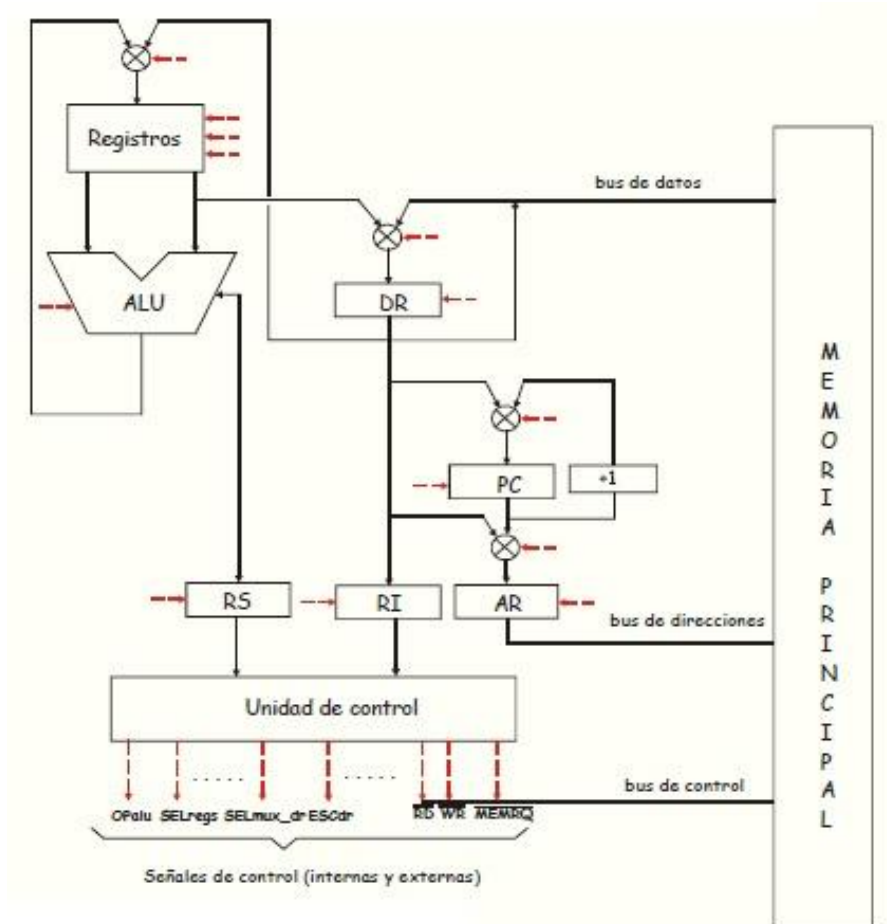


1. **Fase de fetch:** Fase de búsqueda de la instrucción, la unidad de control ordena la lectura de la memoria principal, carga la instrucción en el registro de instrucción (IR) e incrementa el contador de programa (PC). Todas las instrucciones del juego de instrucciones tienen fetch y es igual para todas. Tiene una duración de varios ciclos de reloj, que dependerá de la máquina que se esté utilizando.
2. **Decodificación:** Es el análisis de la instrucción. Planteamos un ciclo de reloj para esta instrucción.
3. **Fase de ejecución:** Ordena mediante las señales de control la ejecución de la instrucción. Como máximo se hace la búsqueda de operandos, se hace la operación completa y el almacenamiento de los resultados.

Los ciclos de reloj que dure la fase de ejecución dependerá de la instrucción, de la máquina, etc. Puede haber instrucciones que duren cero ciclos de reloj en su fase de ejecución, por ejemplo, una instrucción condicional, donde se comprobaría dicha condición en la fase de decodificación y, de no cumplirse, no habría fase de ejecución. La instrucción se ejecuta pero sin su fase de ejecución.

Las acciones dentro de las diferentes fases son ordenadas por el procesador y, dentro de

él, por la unidad de control.



Cuando en los pasos se pone $DR \leftarrow M(AR)$; en realidad el computador hace: MReq, RD, EsperaMemoria, $DR \leftarrow \text{BusDatos}$.

La lectura de MP de otras palabras de instrucción no es un fetch. Si la instrucción tiene más palabras, se realizan operaciones similares al fetch. El PC, después de hacer el fetch, apunta a la siguiente palabra de la instrucción. El PC se incrementa en una palabra si es de direccionamiento a palabra y en 4 bytes si es direccionamiento a byte.

Hay dos hilos en el computador que limitan las operaciones del procesador. En el procesador hay un elemento que determina el ciclo de reloj, este está determinado por la tecnología de la ALU.

En un computador hay diferentes tipos de instrucciones dentro de su juego de instrucciones. No todas las instrucciones dentro del juego de instrucciones pueden estar en los programas de usuario, ya que son del sistema operativo. El computador trabaja en dos modos diferentes de computación: **Modo usuario** o **modo supervisor** (privilegiado).

- *Modo usuario*: sólo se pueden ejecutar un subconjunto de instrucciones de los juegos de instrucciones ya que la máquina no puede ejecutarlas todas por no tener acceso a ellas.
- *Modo supervisor*: es aquí cuando el sistema operativo puede ejecutar todas las

instrucciones y tiene acceso a todos los componentes de la máquina. Sin que el usuario lo note, el SO hace muchas cosas por él.

Dirección		Lenguaje ensamblador
0	Load r1	ld r1, /1000
1	1000	
2	Sub r1 r1 r2	sub r1, r1, r2
3	Store r1	st r1, /1200
4	1200	
5	Salto si z	jmpz /50
6	50	
1000	00000...010	2
1200	00000...101	5

La instrucción de carga de un dato en un registro.

`ld .R1, /1000` $\Leftrightarrow R1 \leftarrow M(1000)$ $1000 = M(PC)$

C.O	R1	
10111...	0001	XXX

No cabe en una sola palabra como vemos en la distribución de la memoria en la imagen superior. También podemos saber si entran en una palabra viendo el tipo de instrucción, o si en ella contiene los símbolos /, \$, #. Lo que se encuentra a la derecha va en otra palabra.

Los pasos que se realizan.

1. Fetch

- $AR \leftarrow PC$; (contiene la dirección de donde está la instrucción).
- Leer MP: $DR \leftarrow M(AR)$; (DR se encuentra la instrucción, no toda porque esta tiene dos palabras).
- $PC \leftarrow PC + 1$; (aumenta el PC apuntando a la siguiente palabra)
- $RI \leftarrow DR$

2. Cargamos la segunda palabra de la instrucción.

- $AR \leftarrow PC$; (contiene la dirección de donde estar la dirección del dato).
- Leer MP: $DR \leftarrow M(AR)$; (DR esta la direccion donde se encuentra el dato, 1000).

- $PC \leftarrow PC + 1$; (aumenta el PC apuntando a la siguiente instrucción)
- 3. Después del fetch viene la decodificación.
 - $AR \leftarrow DR$; (AR tiene la dirección de 1000)
 - Leer MP: $DR \leftarrow M(AR)$; (lee la memoria en la palabra 1000, y pasa el dato a DR)
- 4. A continuación viene la fase de ejecución. Se lee la memoria principal y el contenido se lleva al registro de datos a través del bus de datos. Ahora mismo, en DR está lo que había en la dirección de memoria, que en este caso era otra dirección de memoria (1000).
 - $R1 \leftarrow DR$ (Este lleva el dato al registro).

La instrucción de resta.

$SUB \ .R1, \ .R1, \ .R2 \Leftrightarrow R1 \leftarrow R1 - R2$

Pasos para la resta (Pseudocódigo de la estructura del computador):

1. Fetch
 - a. AR: Carga la dirección de lo que se va a leer con el contenido del PC.
 $AR \leftarrow PC$
 - b. Leer MP (leer instrucción):
 $DR \leftarrow M(AR)$; guarda en DR los datos que haya en la posición de memoria que está guardada en el AR
 $PC \leftarrow PC + 1$; aumenta el PC en uno para que apunte a la siguiente instrucción
 $IR \leftarrow DR$; pasa los datos del DR al IR
2. Decodificación.
3. Fase de ejecución.
 $R1 \leftarrow R1 - R2 \Leftrightarrow SUB \ .R1, \ .R1, \ .R2$
 Actualizar SR; SR tiene el resultado de la última operación lógica que ha realizado la ALU

La instrucción de suma.

$add \ r1, \ /1000 \Leftrightarrow r1 \leftarrow r1 + M(1000) \qquad 1000 = M(PC+1)$

En la estructura del computador en el que estamos haciendo estos ejemplos, no podemos hacer esta instrucción porque la UC no puede utilizar los otros registros que no sean el r1. En esta máquina habría dos soluciones:

1. Física: Poner un multiplexor a la entrada de una entrada de la ALU y conectar al multiplexor la entrada del banco de registros y una del DR y, con una señal de control nueva, la Unidad de Control podría seleccionar cuál de las dos queremos que entre a la ALU

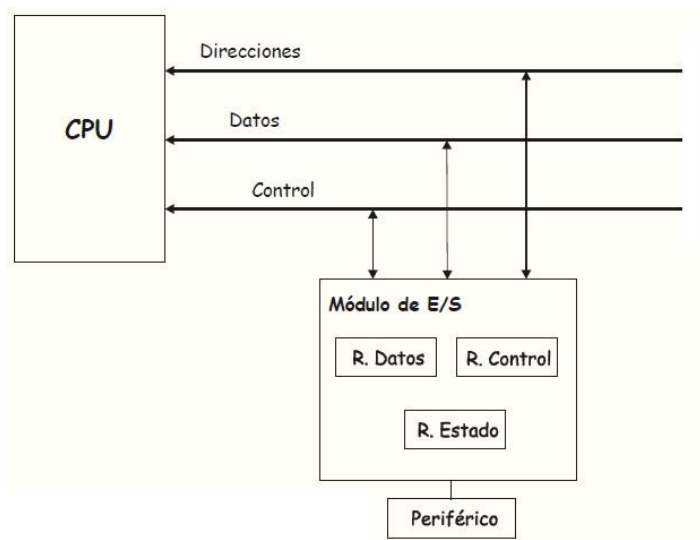
2. Física: Añadir al banco de registros unos registros transparentes para que la unidad de control los pueda utilizar para almacenar temporalmente la información de memoria.
3. Software: antes de hacer esta instrucción, tendríamos que hacer otra que guardará el contenido de la dirección de memoria /1000 en un registro (ld r2, /1000) y luego sumar el registro r1, con el r2 (add r1, r2).

Suponiendo que la realizar la solución 1 y cableamos, las operaciones a realizar por el computador sería:

1. Fetch
 - a. AR: Carga la dirección de lo que se va a leer con el contenido del PC.
 $AR \leftarrow PC$
 - b. Leer MP (leer instrucción).
 $DR \leftarrow M(AR)$; guarda en DR los datos que haya en la posición de memoria que esta guardada en el AR
 $PC \leftarrow PC+1$; aumenta el PC en uno para que apunte a la siguiente instrucción
 $IR \leftarrow DR$; pasa los datos del DR al IR
2. Leer MP (leer registro).
 - a. AR: Carga la dirección de lo que se va a leer con el contenido del PC.
 $AR \leftarrow PC$
 - b. Leer MP (leer instrucción).
 $DR \leftarrow M(1000)$; guarda en DR los datos que haya en la posición de memoria que esta guardada en el AR
 $PC \leftarrow PC+1$; aumenta el PC en uno para que apunte a la siguiente instrucción
3. Leer MP (leer memoria).
 - a. AR: Carga la dirección de lo que se va a leer con el contenido del PC.
 $AR \leftarrow PC$
 - b. Leer MP (leer instrucción).
 $DR \leftarrow M(1000)$; guarda en DR los datos que haya en la posición de memoria que esta guardada en el AR
 $PC \leftarrow PC+1$; aumenta el PC en uno para que apunte a la siguiente instrucción
4. Decodificación.
5. Fase de ejecución.
 $R1 \leftarrow R1 - DR \Leftrightarrow \text{add } r1, /1000$
 Actualizar SR; SR tiene el resultado de la última operación lógica que ha

realizado la ALU

Unidad de Entrada/Salida



La unidad de entrada-salida está conectada al procesador por los buses de direcciones, datos y control que también son usados por la memoria principal. Es la encargada de realizar el intercambio de información y la conexión entre el procesador y los periféricos. La unidad de entrada-salida está formada por módulos de entrada-salida, que son controladores de los distintos periféricos. Los periféricos son todo lo que no es ni el procesador ni la memoria principal, incluida la memoria secundaria donde están los programas que se van cargando.

Ningún periférico es esencial para la estructura básica del computador.

Los dispositivos periféricos y el procesador tienen características muy distintas y también entre ellos son muy diferentes, tanto a nivel de velocidad como a nivel de la información que manejan. Estos módulos de entrada salida ocultan al procesador las diferentes características de los periféricos y son los encargados de la comunicación con los periféricos.

Dos problemas básicos que deben resolverse para realizar esa comunicación entre el procesador y los periféricos:

- Se conoce como “direccionamiento de los dispositivos periféricos”. Se trata de seleccionar el dispositivo con el que se va a establecer la comunicación.
- Transferencia E/S : se refiere a cómo hacer el intercambio de información entre el periférico seleccionado y el resto del computador.

Direccionamiento de los dispositivos:

Se hace asignando a cada registro del módulo de E/S una dirección (puerta de entrada).

Esta dirección se selecciona a través del bus de direcciones, igual que se hace con las palabras de memoria principal. Una vez direccionado, la transferencia se realiza a través del bus de datos.

Esta asignación de direcciones de los dispositivos se puede realizar de dos maneras:

- Mapa de E/S y Memoria principal común: quiere decir que ambos tienen el mismo

conjunto (mapa) de direcciones. En general, estas direcciones están protegidas, no pueden usarlas cualquier usuario, por lo que se usa la misma estructura de memoria para E/S y memoria principal.

No pueden usarse las mismas direcciones ya que el mapa es común.

La diferencia entre una transferencia de E/S y Memoria Principal es mediante la direcciones distintas.

No es necesario usar señales de control distintas ni instrucciones distintas para operaciones de E/S u operaciones de memoria R/W.

Se pueden utilizar las instrucciones de LD (LOAD, cargar registro) y ST (STORE, almacenar en memoria).

- Mapa de E/S y Memoria principal separado: En este caso, al tratarse de mapas de memoria separados, puede usarse la misma dirección para seleccionar un módulo de E/S o para un acceso a la memoria principal. Si las direcciones son las mismas, diferenciamos las instrucciones porque las señales de control son distintas y las instrucciones son específicas.

i. Señales de control diferentes

MEMRQ → petición de Mp

IOREQ → petición de E/S

- ii. Instrucciones específicas para las instrucciones de E/S, que serán IN (desde el periférico al procesador) y OUT (desde el procesador al periférico). Las de Mp siguen siendo LD y ST.

Modos de realizar la operación de transferencia de E/S:

Se refiere a cómo hacer el intercambio de información entre el periférico seleccionado y el computador. Hay distintos modos de realizar la operación de transferencia de E/S.

- E/S programada

Es la CPU la que controla todo el proceso de entrada salida.

La **CPU comprueba el estado del periférico**: si está preparado, envía o recibe el dato y vuelve a comprobar el estado del periférico, y así sucesivamente, es decir, el procesador comprueba continuamente el estado de los periféricos.

El principal inconveniente de este modo es la pérdida de tiempo (se queda en espera activa) del procesador al comprobar el estado de los periféricos, ya que no puede hacer otra cosa y esta operación tarda mucho tiempo en completarse (aun siendo solo 3 instrucciones) aproximadamente 10ms ($10\text{ms} = 10000\mu\text{s} = 10^6\text{ns}$), lo que serían 10 millones de operaciones que podría realizar el ordenador (1 operación tarda 1ns). El estado de dichos periféricos dependerá del tipo que sean. Estas operaciones se realizan mediante lectura y escritura de los registros.

- E/S por interrupciones

Es el **periférico el que avisa** a la CPU, a través de su módulo de E/S, de que está preparado para una transferencia. La CPU ordena la transferencia y se encarga de realizarla.

La transferencia se hace mediante los siguientes pasos:

1. Interrumpe la ejecución del programa en curso.
2. Salva el estado de la máquina.
3. Ejecuta una **rutina de tratamiento de interrupción** en la que da servicio a ese periférico en concreto. Esta rutina la ejecuta el SO en modo privilegiado.
4. Restaura el estado.
5. Continúa la ejecución del programa interrumpido.

La CPU comprueba si hay petición de interrupción al final de cada instrucción.

La interrupción se trata como un salto condicionado.

La ventaja es que el tiempo que ocupa el procesador en la operación E/S es menor que la E/S programada porque no tiene que observar el periférico.

- E/S por acceso directo a memoria (DMA)

En este modo, la CPU sólo se encarga de iniciar la operación, es decir, de ordenar la transferencia, **pero no ejecuta instrucciones para producirla**. Es el módulo del periférico el que accede directamente a memoria. Cuando el módulo de E/S ha terminado, se lo indica al procesador.

Ocupa menos tiempo porque el procesador está libre para realizar otro tipo de operaciones.

Hay que implementar mecanismos que controlen la compartición de los buses.

Software de sistemas

Compiladores y ensambladores

El compilador genera, a partir de un lenguaje de alto nivel, un programa en lenguaje máquina. El lenguaje ensamblador es el más cercano al lenguaje máquina pero con símbolos especiales (mandatos, instrucciones) que representan directamente una combinación de 0 y 1 en el lenguaje máquina.

Cargadores (bootstrap)

Se encarga de transferir el programa a ejecutar a la Memoria Principal.

Sistema operativos

El sistema operativo es el conjunto de programas que realiza la gestión de recursos (CPU, MP, E/S) de la máquina pero además oculta al procesador la complejidad de los periféricos (crea una visión irreal que es más comprensible por el usuario) y protege los recursos. Se ejecuta en modo supervisor, privilegiado o root.

Parámetros característicos

- Ancho de palabra: el número de bits que maneja en paralelo el computador. suele coincidir con el tamaño de los registros y el tamaño del bus de datos. Actualmente son de 64 bits, pero antes tenias de 8, 16 y 32.
- Tamaño de memoria: generalmente expresado en bytes y sus correspondientes múltiplos.

IEC prefix		Representations				Customary prefix	
Name	Symbol	Base 2	Base 1024	Value	Base 10	Name	Symbol
kibi	Ki	2 ¹⁰	1024 ¹	1 024	≈1.02 × 10 ³	kilo	k, K
mebi	Mi	2 ²⁰	1024 ²	1 048 576	≈1.05 × 10 ⁶	mega	M
gibi	Gi	2 ³⁰	1024 ³	1 073 741 824	≈1.07 × 10 ⁹	giga	G
tebi	Ti	2 ⁴⁰	1024 ⁴	1 099 511 627 776	≈1.10 × 10 ¹²	tera	T
pebi	Pi	2 ⁵⁰	1024 ⁵	1 125 899 906 842 624	≈1.13 × 10 ¹⁵	peta	P
exbi	Ei	2 ⁶⁰	1024 ⁶	1 152 921 504 606 846 976	≈1.15 × 10 ¹⁸	exa	E
zebi	Zi	2 ⁷⁰	1024 ⁷	1 180 591 620 717 411 303 424	≈1.18 × 10 ²¹	zetta	Z
yobi	Yi	2 ⁸⁰	1024 ⁸	1 208 925 819 614 629 174 706 176	≈1.21 × 10 ²⁴	yotta	Y

- Frecuencia de reloj: determina a qué velocidad se van a producir los eventos hardware del computador. Ahora estas se encuentran más estancadas en velocidades, que rondan los 2,8GHz - 3,2GHz.
- Duración de las operaciones: invirtiendo nos dará la ¿frecuencia?. Expresadas en múltiplos de segundo (1s = 10³ms = 10⁶μs = 10⁹ns = 10¹²ps = 10¹⁵fs).
- Capacidad de cómputo o velocidad: Indica la productividad, el throughput. Se utiliza para comparar los grandes computadores. Trabajo útil por unidad de tiempo. Número de instrucciones que se ejecuta por unidad de tiempo. Hay dos unidades

que se han utilizado a lo largo de la historia, el MIPS (millon de instrucciones por segundo, MI/s), el MFLOPS (millones de operaciones en coma flotante por segundo), specint y specfp.

- Ancho de banda (caudal): Es la cantidad de información que es capaz de transmitir un bus o una unidad de entrada o salida o en general la que es capaz de tratar una determinada unidad en cada segundo. Puede expresarse en bytes o en bits

Ejemplo

Examen Enero 2013 Recuperación

PROBLEMA 1: Indique, justificando su respuesta si las siguientes afirmaciones son verdaderas o falsas:

- A. La E/S por interrupciones es la que ofrece mejor rendimiento en la transferencia de un bloque de datos a un periférico.

FALSO

Respuesta: No es la mejor porque existe otro método, que es el acceso directo a memoria, que proporciona mejor rendimiento porque el procesador no se ocupa de la transferencia.

- B. La unidad de control no necesita como entrada el PC.

VERDADERO

Respuesta: Porque la unidad de control solo necesita como entrada el registro de estado e instrucción

- C. El SR es un registro transparente al usuario ya que este no tiene por qué utilizarlo

FALSO

Respuesta: No es un registro transparente, puede aparecer en la especificación de las instrucciones. Por lo tanto es un registro específico.

- D. El modelo Von Newmann se basa en una única memoria para almacenar tanto datos como instrucciones, por lo que se podría, erróneamente, intentar ejecutar un dato.

VERDADERO

Respuesta:

- E. El PC es un registro transparente al programador.

FALSO

Respuesta: El PC es un registro visible al programador de propósito específico.

- F. El registro de instrucción es un registro de propósito específico que contiene la siguiente instrucción a ejecutar.

FALSO

Respuesta: No es de propósito específico, es un registro transparente.

- G. La arquitectura Von Newmann tiene almacenados los datos separados de las instrucciones en memorias distintas.

FALSO

Respuesta: Se almacenan datos e instrucciones en la misma memoria.

- H. El registro de direcciones de memoria es un registro de propósito general, que puede contener tanto direcciones como datos.

FALSO

Respuesta: Un registro de instrucciones no puede contener datos y además es transparente.

TEMA 2 - PROGRAMACIÓN EN ENSAMBLADOR

Introducción

El procesador que veremos será el MOTOROLA 88110. Las instrucciones que se usen en un determinado procesador son específicas para ese procesador y no pueden usarse en otro.

Veremos un estándar IEEE (mucho más amplio que el lenguaje de lo que realmente utiliza el procesador), no utilizaremos el propio del 88110.

El lenguaje ensamblador con las estructuras de datos y las subrutinas.

Lenguaje Máquina

- El programa está compuesto por datos e instrucciones almacenados en memoria.
- **Instrucción máquina:** es la función básica elemental que puede ejecutar un computador.
- Son cadenas de 1s y 0s y particulares de cada computador.
- **Las operaciones aritméticas** tienen un número fijo de operandos, realizan una única y sencilla función y **son autocontenidas**, es decir, contienen todo lo necesario para su ejecución.

Juego de instrucciones

- Conjunto de instrucciones que ejecuta directamente el computador.
- La codificación de las instrucciones debe encajar en pocos formatos.
- **Formato de instrucción:**
 - Código de operación.
 - Operandos (direcciones).

Tipos de datos

Los tipos de datos están almacenados en memoria. La cantidad de información que se transfiere a la memoria es una palabra (de tamaño dependiendo del procesador). Hay muchos datos que son de tipo carácter, que se puede representar con 1 byte (8 bits), porque hay mucho intercambio de información con los periféricos y se usa mucho texto.

Los tipos de datos que maneja una instrucción son:

- Palabras: Es el tamaño privilegiado del computador (4 bytes).
- Medias palabras (2 bytes). En el caso necesitemos menos espacio.
- Bytes: Cadenas de caracteres (1 byte).

El acceso a memoria siempre es a nivel de palabra. Cuando se accede a memoria y se pide que lea, ella devuelve una palabra.

El acceso a memoria se puede hacer de dos maneras (también explicado en el tema 1 de introducción):

- Direccionamiento a nivel de byte: es como si todas las palabras de la memoria estuvieran seguidas en una fila. Es más versátil, ya que se puede acceder más

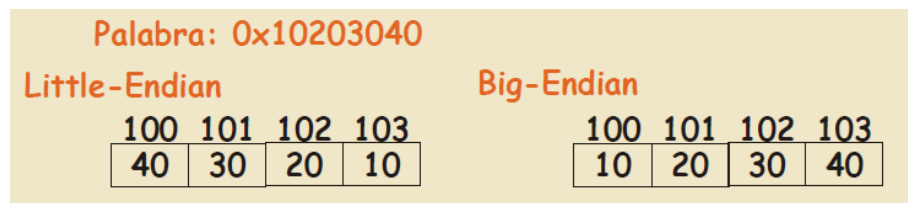
directamente a lo que uno quiere. Hay que tener cuidado con el tamaño de la palabra (32 bits - 4 bytes, por ejemplo) pues la siguiente palabra tendrá la dirección $X+4$ (Ésto sólo pasa si la palabra en cuestión es de 4 Bytes, el ancho de palabra PUEDE VARIAR, por lo tanto sería $X+Y$)

- Direccionamiento a palabra: Cada palabra tiene una dirección. Por tanto, como la memoria está organizada en palabras del tamaño de la palabra, es este caso 32 bits, la palabra 0 tendrá 32 bits.

Alineamiento a palabra: La dirección para el acceso a palabra debe ser múltiplo del tamaño de la misma, especialmente para el direccionamiento a byte. Para saber si está alineada a palabra basta con mirar que los dos últimos bits menos significativos sean cero los dos.

Ordenación de bytes de una palabra en memoria:

- Little-Endian: Byte menos significativo de una palabra en la dirección menos significativa.
- Big-Endian: Byte menos significativo de una palabra en la dirección más significativa.



Antiguamente se hacía direccionamiento a palabra, pero evolucionó por la velocidad a la que se podía hacer operaciones aritméticas más rápidas.

Modos de direccionamiento

- Forma en la que accedemos a los operandos.
- Forma que tiene la instrucción de indicar en qué lugar está el objeto.
- Los **objetos importantes de una instrucción** son: datos (Operando y resultado) e instrucciones
- La **dirección o lugar donde residen dichos objetos** puede ser en: Instrucciones, Registros o Memoria.

Tipo de direccionamiento

Tipo de direccionamiento

Inmediato

IEEE 694

#dato

LD .R1, #5

R1 ← 5

ADD .R1, #4

R1 ← R1+4

Directo

Absoluto

a Registro

.Reg

LD .R1, #5

R1 ← 5

ADD .R1, #5

R1 ← R1 + 5

a Memoria

/dir

LD .R1, /1000

R1 ← M(1000)

Relativo			
a Registro base	#desp[.Reg]		
	LD .R1, #5[.R2]	R1	$\leftarrow M(R2+5)$
	LD .R1, #-5[.R2]	R1	$\leftarrow M(R2+(-5))$
a PC	\$desp		
	BR \$5		
a Registro índice			
	preincr	#desp	[++.RI]
	predecc	#desp	[--.RI]
	postincr	#desp	[.RI++]
	postdecc	#desp	[.RI--]
Indirecto			
a Registro	.Reg	M(Reg)	
a Memoria	/dir	M(M(dir))	
Implícito			

- **Direccionamiento inmediato:** El objeto está contenido en la propia instrucción.
- **Direccionamiento directo:** El objeto no está contenido en la propia instrucción. La instrucción contiene el lugar (dirección) donde está almacenado el objeto.
 - El **direccionamiento se considera absoluto** si la instrucción contiene la dirección completa del objeto.
 - **A registro:** El objeto del direccionamiento está contenido en un registro. La instrucción contiene el registro que contiene el objeto del direccionamiento.
 - **A memoria:** El objeto del direccionamiento está contenido en una dirección de memoria. La instrucción contiene la dirección completa de memoria que contiene el objeto del direccionamiento.
 - El **direccionamiento relativo**, se considera si la instrucción contiene la dirección del objeto de forma parcial. Este tipo de direccionamiento son siempre a memoria. Pueden ser: Relativo a registro base, relativo a PC, registro índice.
 - A registro base:
 - La dirección de memoria viene especificada en dos partes:
 - Registro Base: Registro de propósito específico o general que contiene una dirección a memoria.
 - Desplazamiento: Valor entero con signo.
 - La dirección efectiva se calcula: $Dir_Efectiva = Registro_Base + Desplazamiento$
 - El registro base se carga una dirección de memoria que contiene un conjunto de datos a los que se accede conociendo su posición relativa frente al comienzo de dicha zona: Estructuras de datos.
 - El rango de direcciones al que se puede acceder está limitado por el tamaño del desplazamiento.
 - La suma del registro base con el desplazamiento se hace complemento a dos con extensión de signo.

- A PC:
 - Es un direccionamiento relativo a registro base en el que el registro base es el PC.
 - El objeto de este direccionamiento siempre es a instrucciones, ya que el PC apunta a la siguiente instrucción. Se podría hacer con datos, pero es más complicado.
 - Permite alcanzar instrucciones “cercanas” a la que se está ejecutando.
 - El desplazamiento tiene que ser múltiplo de 4 si es direccionamiento a byte. Y puede ser muy grande. por estructura de la instrucción, ya que el tamaño del objeto es el tamaño de la palabra meno el código de la operación y esos son muchos bits, que pueden tener un rango muy grande a ser en complemento a dos.
 - Ejecución de saltos “cortos”: \$desp BR \$5
- A registro índice:
 - Es un direccionamiento relativo a registro base en el que el registro base se modifica.
 - Preincremento #desp [++.Ri]:
 LD.B .R1,#8[++.R7] R7←R7+1 R1←M(R7+8)
 LD .R1,#8[++.R7] R7←R7+4 R1←M(R7+8)
 - Predecremento#desp [--.Ri]:
 LD.B .R1,#8[--.R7] R7←R7-1 R1←M(R7+8)
 LD .R1,#8[--.R7] R7←R7-4 R1←M(R7+8)
 - Postincremento
 - Postdecremento
 - El tamaño del incremento/decremento es igual al tamaño del objeto transferido, si es byte pues mas 4, si es palabra más 1.
 - Útil para recorrer vectores y matrices.
- Direccionamiento indirecto: Se acuerda donde le vas a dejar el dato, por eso no es necesario. **Registro acumulador.**
 - A registro:
 - Contiene la especificación del registro que contiene la dirección de memoria.
 ADD .R1, [.R2] R1 <- R1+M(R2)
 BR [.R2] PC <- R2
 - A memoria:
 - ADD .R1, [/1000] R1 <- R1+M(M(/1000))
- Direccionamiento implícito:
 - Ni la dirección ni el objeto está contenido en la instrucción. Antiguo, antes del 1980.
 ADDA .R1 A <- A+R1

Ejemplo

Sabiendo que en los registros generales R1 y R2 tienen las direcciones de memoria, 0, 1000 respectivamente y que en las direcciones de memoria 1004 hay un 4, en la 1005 hay

un 5 y en la 1006 hay un 6, diga qué contienen los registros R1 y R2 cuando hacen las siguientes instrucciones.

a) LD .R1, #5[.R2++]

Tenemos un postincremento, por tanto hacemos primero todo y luego incrementamos.

$$R1 \leftarrow M(1000+5) = 5$$

$$R2 \leftarrow 1000+1 = 1001$$

b) LD .R1, #5[++.R2]

Tenemos un preincremento, por tanto lo hacemos primero y luego el resto.

$$R2 \leftarrow 1000+1 = 1001$$

$$R1 \leftarrow M(1001+5) = 6$$

c) LD .R1, #5[--.R2]

Tenemos un predecremento, por tanto lo hacemos primero y luego el resto.

$$R2 \leftarrow 1000-1 = 999$$

$$R1 \leftarrow M(999+5) = 4$$

d) LD .R1, #5[.R2--]

Tenemos un postdecremento, por tanto hacemos primero todo y luego decrementamos.

$$R1 \leftarrow M(1000+5) = 5$$

$$R2 \leftarrow 1000-1 = 999$$

Juego de instrucciones

Transferencia de datos

MOVE	R -> R M -> M	MOVE orig, dest	MOV /1000, /2000 MOV .R4, .R5 R1->R4 MOV #4[.R5], #8[.R4] - M(R5+4)->M(R4+8)
LOAD	R <- M inm	LD dest, orig	LD .R1, #4[.R4] R1<-M(R4+4) LD .R1, /1000 R1<-M(1000) LD .R1, #1000 R1<-1000
STORE	R -> M inm	ST orig, dest	ST .R1, #4[.R4] R1->M(R4+4) ST .R1, #4[.R4] R1->M(1000)
PUSH		OUT orig, dest	PUSH .R SP<-SP-4 R1 <- M(SP)

POP			POP .R1 R1 <- M(SP) SP<- SP+4
-----	--	--	----------------------------------

Que la mayoría de las transferencias se realizan de destino a origen (dest, orig) excepto MOVE, STORE, OUT que son de origen a destino, son excepciones.

- SP (Stack Pointer) -> net dato; dir decrec
 - PUSH .R1 \equiv predecr SP
 - ST .R1, [--.SP] (si no hay push)
 - POP .R1 \equiv postincr SP
 - LD .R1, [.SP++] (si no hay pos)
- SP -> net dato; dir crec
 - PUSH .R1 \equiv preincr SP
 - ST .R1, [++.SP] (si no hay push)
 - POP .R1 \equiv postdecr SP
 - LD .R1, [.SP--] (si no hay pos)
- SP -> 1º hueco; dir crec
 - PUSH .R1 \equiv postincr SP
 - ST .R1, [.SP++] (si no hay push)
 - POP .R1 \equiv predecr SP
 - LD .R1, [--.SP] (si no hay pos)
- SP -> 1º hueco; crec dec
 - PUSH .R1 \equiv postdecr SP
 - ST .R1, [.SP--] (si no hay push)
 - POP .R1 \equiv preincr SP
 - LD .R1, [++.SP] (si no hay pos)

Salto (BRANCH) o bifurcaciones

Incondicionales: Direccionamiento a memoria siempre.

BR (IEEE):

EL objeto es la siguiente instrucción a ejecutar.

BR /1000; PC <- 1000

BR \$10; PC <- PC+10

BR [.R1]; PC <- R1 direccionamiento indirecto

BR #4[.R4]; PC <- M(R4+4)

---> BR #1000 imposible porque le pasas un dato en vez de una dirección de memoria, tiene que ser una direccionamiento inmediato.

---> BR .R1 Se necesita la siguiente instrucción y r1 no puede tener instrucciones, sólo dirección y datos.

jmp (88110):

Condicionales (Bc):

Formato: Bcc dir

cc: si cc=1 se ejecuta la siguiente operación.

dir: es una dirección a memoria donde se encuentra la siguiente instrucción a ejecutar.

BC:

salto si c=1, si no sigo en secuencia.

BC \$10 PC <- PC+10

BNC:

salto si c=0, si no sigo en secuencia.

BC /1000 PC <- 1000

BZ:

salto si z=1, si no sigo en secuencia.

BC #4[R7] PC <- R7+4

BNZ:

salto si z=0, si no sigo en secuencia.

BC [.R8] PC <- R8

BV:

salto si v=1, si no sigo en secuencia.

BNV:

salto si v=0, si no sigo en secuencia.

BP:

salto si s=0, si no sigo en secuencia.

BN:

salto si s=1, si no sigo en secuencia.

Con retorno

Siempre hay que hacer un CALL y luego un RET

CALL: Con retorno

El mejor sitio para guardar los retornos es en una pila -> PUSH .PC

Si hay una pila, hay problemas, porque guarda la DR (dirección de retorno) encima de los parámetros que necesitaría la subrutina. Hay dos soluciones: sacar de la pila (POP) o con direccionamiento relativo al puntero de pila (#12[.SP])

CALL /1000 SP <- SP-4; M(SP) <- PC; PC <- 1000

Pero también se puede hacer de propósito general. Te lo dice en el juego de instrucciones, suele estar implícito en la instrucción. Pero para hacer más de dos llamadas anidadas hay que guardar la primera dirección de retorno de la primera en la pila, porque si no lo sobrecribiría. En el 88110

CALL /1000 R1 <- PC; PC <- 1000

RET: Sin retorno

Si se hace por pila:

RET PC <- M(SP); SP <- SP+4;

Por registro de propósito general:

En el 88110 no tiene RET porque la máquina que dejan la dirección en un registro de propósito general directamente el RET se hace con

BR [.R1], o en el caso del 88110 con jmp (r1).

RET PC <- R1 -> BR[.R1]

Todas las subrutinas tienen que acabar con un RET.

No lleva direcciones asociadas. Vuelva un nivel para arriba.

Aritméticas

DIAPPOSITIVA 26

Modifican el registro de estado.

Determinan el número de operandos del procesador y el modelo de ejecución del procesador.

Si solo tienen dos direcciones la solución de la operación se guardará en la primera dirección puesta en la instrucción.

- La suma:

ADD .R1, .R2 R1 <- R1+R2; mod. flags

- La resta: no se guarda en ningún registro ya que se guarda en el registro de estados.

SUB .R1, .R2 R1 <- R1 - R2;

- La comparación es una instrucción muy utilizada. hace una resta, no se almacena el resultado en ningún sitio pero si que modifica los flags de estado.

CMP .r1,.r2; Bz \$; // salta si los dos registros son iguales.

- La suma con acarreo:

ADDC .R1, .R2 R1 <- R1+R2; mod. flags

- Incremento y decremento: modifican el registro de estado

INC .R1 ADD .R1, #4; R1 <- R1+1; mod. flags

DEC .R1 SUB .R1, #4; R1 <- R1-1; mod. flags

Lógicas

Trabaja bit a bit entre los dos operandos de la instrucción.

Modifica los flags de estado.

Para poner a 0 un registro tenemos que hacer un XOR .R1, .R1

Máscara con un AND y así sabemos los múltiplos. Ej: múltiplo de 16 AND .R1, #15

Si es par: AND .R1, #H'80000000 por que eso que esta en HEX, en binario en 32 bits es 1 y como es Little Endian pues se carga al revés.

Desplazamientos

Pueden ser hacia la derecha (SHR) o hacia la izquierda (SHL), aritméticos o lógicos y circulares o ...

SHL .R1 -> desplazamiento a la izquierda -> es como multiplicar por 2

SHR .R1 -> desplazamiento a la derecha -> es como dividir por 2

ROL .R1

ROR .R1

ROL .R1

RORC .R1

De bit Lógicas

CLR.I #X, .RY	Poner a 0 el bit X de RY
SET.I #X, .RY	Poner a 1
TEST.I #X, .RY	Z - NOT(RY(3)) -> salto concional

HALT

WAIT

Ejercicios

Ejercicios y Problemas (Instrucciones y direccionamientos) ([inst_dir_12-13.pdf](#))

La mayoría de los ejercicios de esta hoja de problemas están resueltos en otro doc en la teoría del tema dos llamado igual que el pdf.

8

Problema

Para un computador de dos direcciones con modelo de ejecución registro-registro con palabra de 32 bits y direccionamiento a nivel de byte que dispone únicamente de direccionamiento inmediato, directo a registro e indirecto a registro, realice los programas correspondientes a las instrucciones que se muestran a continuación para el computador indicado. Si es necesario utilice los registros RT1, RT2 y RT3 como registros auxiliares.

1. ADD /1000, /2000
2. CALL #4[.R7]
3. SUB .R1, [#4[.R7+
4. +]] Solución

Tenemos un computador que tiene estas características:

- Modo de ejecución: Registro - Registro
- Tamaño de palabra: 32 bits
- Acceso a memoria: Byte
- Modos de direccionamiento: inmediato (#), directo a registro (.rX), indirecto a registro ([.rX])

Con esas características tenemos que ver si las instrucciones a continuación son

1. ADD /1000, /2000
2. CALL #4[.R7]
3. SUB .R1, [#4[.R7++]]

Otros

1.

BR #1000 Es un salto y el modo de direccionamiento es inmediato. Es incorrecto, porque lo que tiene que venir en el campo de dirección es la dirección a la que vas a saltar

BR /100 $PC \leftarrow 100$. Se carga como si fuese un dato inmediato

BR .R1 Las instrucciones no pueden estar almacenadas en un registro.
INCORRECTO

BR [.R1] Un direccionamiento indirecto a registro es lo mismo que un direccionamiento relativo a registro base con desplazamiento. Lo que está en el registro R1 lo guardo en el contador de programa. $PC \leftarrow R1$

BR #10 [.R1] En el contador de programa añado 10 + Registro R1 $PC \leftarrow 10 + R1$

BR \$10 El \$ indica el desplazamiento relativo al PC, está en memoria en dirección $PC \leftarrow PC + 10$

BR [#10[.R1]] $PC \leftarrow M(R1 + 10)$

2. Un computador tiene como únicos modos de direccionamiento permitidos: directo a registro, inmediato y relativo a registro base. Realice en ensamblador IEEE las secuencias de instrucciones equivalentes a las siguientes:

MOVE [#3[.R18]], /1000
BR /1000
BR [#4[.R18]]

LD.R1, #1000 Se carga en R1 el número 1000, que corresponde a una posición de memoria. No se podría usar ADD porque sumaría 1000 al contenido actual de la dirección de memoria R1. Es equivalente a $\#0[.R1]$

LD.R2, #3[.R18] Es equivalente a $\#0[.R2]$

MOVE $\#0[.R2]$, $\#0[.R1]$

BR /1000

LD.R1 #1000

BR $\#0[.R1]$

BR [#4]

LD.R1, #2	;3 bytes	$R1 \leftarrow 240$
LD.R3, #1005	;3 bytes	$R3 \leftarrow 1005 \ 1004 \ 1003$
MOV .R3, .R5	;1 byte	$R5 \leftarrow 1005 \ 1006-1007$
DEC .R1	;1 byte	
MOV #-3[--R3], #-1[.R5++]	;2 bytes	MOV $M(-3+R3) \rightarrow M(-1+R5)$

		M(1001) → M(1004)
		R5++

		MOV M(-3+R3) → M(-1+R5)
		M(1000) → M(1005)
		R5++
BNZ \$-5	;2 bytes	
HALT	;1 byte	STOP
1000 10		
1001 21		
1002 31		
1003 43		
1004 76 21		
1005 78 10		

3. Sea un computador de dos direcciones con modelo de ejecución registro-registro que tiene los siguientes modos de direccionamiento: inmediato, directo a registro y relativo a registro base. Construya los programas equivalentes a las siguientes direcciones:

```
BR [.R4]
BR /1000
ADD [.R10], [#7[.R10]]

LD.R1, #0[.R10]
LD.R2, #7[.R10]
LD.R2, #0[.R2]
ADD .R1, .R2
ST .R1, #0[.R10]
```

4. Indique cuál sería el contenido de las posiciones de memoria y registros modificados por el siguiente fragmento de programa que se ejecuta en un computador de 8 bits. Nota: La instrucción AND no modifica el flag de acarreo. La posición 100 de memoria y siguientes contienen los valor H'F4, H'01, H'10, H'C2.

```
LD .R1, #100
```

R1 ← 400 404 402 403 104

LD .R2, #0		$R2 \leftarrow 04249$
LD .R3, #4		$R3 \leftarrow 43240$
CLEARC	;1 pal	$C \leftarrow 04040$
AND [.R1], #H'80	;2 pal	AND M(100) ^ 80 = 80
		AND M(101) ^ 80 = 00
		AND M(102) ^ 80 = 00
		AND M(103) ^ 80 = 80
ROLC [.R1++]	;1 pal	
ROLC .R2	;1 pal	
DEC .R3	;1 pal	
BNZ \$-7	;1 pal	
HALT	;1 pal	STOP
100 F4 80 00	R1 104	
101 04 00	R2 8	
102 40 00	R3 0	
103 02 80 00	C 1	

1. En un computador con palabras y direcciones de 16 bits, con direccionamiento a nivel de palabra, se ejecuta el siguiente fragmento de código en el que el tamaño de cada instrucción se indica como comentario. Considere que a partir de la dirección de memoria H'1008 se encuentran almacenados respectivamente a los siguientes datos: H'0001, H'0002, H'0003, H'0004 y H'0005. Indique los valores sucesivos que toman los registros y posiciones de memoria afectados por la ejecución.

```
LD    .R1, #100C      ;2 pal      R1 ← 100C 100B 100A 1009 1008
LD    .R2, #1008      ;2 pal      R2 ← 1008 1007 1008 1007 1008
SUB   .R1, [.R2--]    ;1 pal
DEC   .R1              ;1 pal
ST    .R1, #4[++.R2]  ;2 pal
CMP   .R1, .R2         ;1 pal
BNZ   $-6              ;1 pal
MOV   [++.R2], [.R1++] ;1 pal
HALT
```

```
1008 | 04 02
1009 | 02
100A | 03
100B | 04
100C | 05 100A 1008
```

2. Sea un computador con palabras y direcciones de 32 bits, direccionamiento a nivel de byte y ordenamiento little-endian que ejecuta el siguiente fragmento de código en el que todas las instrucciones ocupan una palabra. Las direcciones de memoria 0 a 7 contienen los siguientes valores hexadecimales respectivamente: EF, FE, 00, 00, DF, FD, 00, 00.

Por ser little-endian son 32bits y cada instrucción ocupa 4 bytes

LD	.R2, #H'0008	;	R2 ← 0008 0004 0000 FFFC
LD	.R4, [-- .R2]	;	R4 ← FE EF
CMP	.R4, #0	;	R4 no es 0
			R4 no es 0
BZ	\$8	;	No se ejecuta el salto
			No se ejecuta el salto
CMP	.R2, #0	;	R2 no es 0
			R2 es 0
BNZ	\$-20	;	Se ejecuta el salto
			No se ejecuta el salto
ADD	[.R2--], .R4	;	
HALT			

0000 | ~~EF~~ DE
 0004 | FE
 0008 | 00
 000C | 00
 0010 | DF
 0014 | FD
 0018 | 00
 001C | 00

Indique razonadamente los valores sucesivos que toman los registros y posiciones de memoria que se modifican durante la ejecución de las instrucciones.

Arquitectura 88110

Procesador

Nosotros usaremos un simulador de este procesador.

Es un procesador RISC de los 80 con un juego de instrucciones reducidos con modelo de ejecución Registro - Registro con máquina de 3 direcciones. Todas las instrucciones ocupan una palabra.

La ALU opera en complemento a 2.

El destino siempre es un registro.

El origen puede ser de: un registro + registro, o de registro + inmediato.

Bancos de 32 registros.

r0: siempre cableado a 0

r1: guarda dirección de retorno de subrutina accesible por pares de 64 bits

Para aumentar o decrementar se hace de 4 en 4

En este procesador el Contador de Programa (PC) apunta a la instrucción que se está ejecutando en ese mismo momento. "Es una rareza".

Este procesador no tiene un puntero de pila. Nosotros utilizaremos un registro que nos haga de pila que será el r31.

Puntero de pila será el r30

PSR (Processor Status Register, registro de estados)

bit 12: overflow en operaciones con enteros (para el modo serie) (OVF). Si OVF == 1

→ no se modifica rD

bit 28: acarreo

r30 puntero de pila

r31 puntero marco de pila

Memoria principal

- Almacena: Instrucciones + Datos
- Direccionable a nivel de byte
 - 1 palabra <-- 4 direcciones de memoria
- Bus de direcciones de 32 bits
 - Máxima capacidad (teórica) --> 2^{32} bytes = 4 GB
 - 0x00000000
 - 0xFFFFFFFF
- Capacidad del emulador:
 - 2^{18} direcciones = 2^{18} bytes = 256 KB

0x00000000
0x0003FFFF

Modos de direccionamiento

- Sí tiene:
 - Directo a registro: `.Ri`
 - Inmediato: `#aaaa`
 - Relativo a registro base: `#desp[.Ri]`
 - Relativo a PC: `$xx`
 - Indirecto a registro: `[.Ri]`
- NO tiene:
 - Absoluto: `/dir`
 - Relativo a registro índice: `++, --`
 - Indirecto a memoria: `[/dir]`
 - Relativo a pila: `push / pop`

Direccionamiento directo a registro

`add r1, r2, r3 ; r1<-- r2 + r3`
equivalente en máquina de 2 direcciones, IEEE 694:
`ADD .R7, .R9 ; (add r7, r7, r9)`

Direccionamiento inmediato

- Puede ser con/sin signo, ambos de 16 bits:
 - Con signo: `SIMM16`
 - Sin signo: `IMM16`
- Se puede expresar en decimal o hexadecimal
- Ejemplo:
 - `add`: suma con signo (lo rellena con "1" a la izq hasta los 16 bit)
 - `addu`: suma sin signo (lo rellena con "0" a la izq hasta los 16 bit)

Direccionamiento relativo a registro base

- Ejemplo en formato del estándar IEEE:


```
LD .R1, #13[.R4]      R1 <-- MEM(R4+13)
```
- Ejemplo en formato de 88110 (desplazamiento inmediato):


```
ld r1, r4, 13         r1 <-- MEM(r4+13)
st r1, r4, 13         r1 --> MEM(r4+13)
```
- Ejemplo en formato de 88110 (desplazamiento en registro):


```
ld r1, r4, r10        r1 <-- MEM(r4+r10)
st r1, r4, r10        r1 <-- MEM(r4+r10)
```

Direccionamiento relativo a PC

- El desplazamiento es relativo a la posición que se encuentra esa instrucción en memoria.
- El desplazamiento en la instrucción de salto tiene que ser múltiplo de 4 (los dos últimos bits menos significativos tiene que ser cero), ya que todas las instrucciones ocupan una palabra y la memoria es direccionable a nivel de byte y en una palabra hay 4 bytes. Si no fuera múltiplo podría saltar a la mitad de una instrucción o dato.
- Ejemplo:

- ```
br 7 ; ; PC <-- PC + 7*4
(desplazamiento: 26 bits / 16 bits)
```
- Ejemplo en formato del estándar IEEE:

```
ADD .R7, .R5
BR $desp ; PC <-- et1 + desp
et1: LD .R1, [.R7]
```
  - Ejemplo en formato de 88110:

```
add r7, r7, r5
et0: br D ; PC <-- et0 + 4*D
ld r1, r7, 0
```

### Direccionamiento indirecto a registro

En el 88110 solo existe en los saltos:

- Ejemplo en formato del estándar IEEE:

```
JMP [.R10] ; PC <-- R10
```
- Ejemplo en formato del 88110:

```
jmp (r10) ; PC <-- r10
```

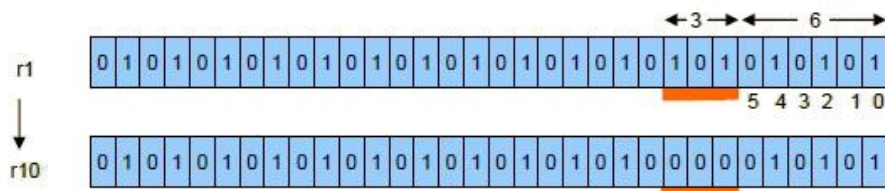
### Direccionamiento campos de bit

En ciertas instrucciones del 88110 se pueden seleccionar:

- bits individuales
- campos de bit

Ejemplo en formato del estándar IEEE / 88110:

```
CLR.I3 .R10, .R1, 6 / clr r10, r1, 3<6>
```



Otro ejemplo (bits individuales):

```
bb0 3, r8, 7 ; Si (bit3 de r8)==0 ==> PC <-- PC + 4*7
```

## Juegos de Instrucciones

Tipos de instrucciones en el 88110:

- Lógicas (or, and, xor, mask)
- Aritméticas (add, sub, addu, subu, muls, mulu, divs, divu, cmp)
- Bifurcaciones (bb0, bb1, br, bsr, jmp, jsr)
- Transferencia (ld, st, ldcr, stcr, xmem)
- Campos de bit (clr, set, ext, extu, mak, rot)
- Coma flotante (fadd, fsub, fmul, fdiv, fcvt, flt, int, fcmp)

Instrucción específica del emulador: stop

### Instrucciones lógicas

or

```
rD, rS1, rS2 r1 <-- r2 OR r3
rD, rS1, IMM16 r1LL <-- r2LL OR IMM16 ; r1LH <-- r2LH
```

Solo se hace la operación con la parte baja de los registros

```
.u r1LL <-- r2LL OR IMM16 ; r1LH <-- r2LH
 Opera con los 16 bits más significativos de rS1
```

```
and rD, rS1, rS2 / IMM16
 .c Hace el complemento a 1 de rS2
 .u Opera con los 16 bits más significativos de rS1
```

```
xor rD, rS1, IMM16
 .c Multiplexor controlado
```

```
mask rD, rS1, IMM16
 .u
```

Ejemplo: del IEEE al 88110

```
LD .R1, #H'90000001
 or r1, r0, 0x0001; r1LL <- 0001 ; r1LH <- 0
 or.u r1, r1, 0x9000; r1LL <- r1LL ; r1LH <- 9000
 ; r1 = H'90000001
NOR .R1
 xor.c r2, r1, r0;
```

Transferencia entre registro or r1, r2, r2;

### Instrucciones aritméticas

```
add suma
 add
 .u sin signo, causan overflow
 .ci opera con acarreo de entrada (bit 28 de PSR)
 .co actualiza el flag de acarreo (bit 28 de PSR)
 .cio equivalente a usar .ci+.co
sub resta
 sub causa overflow
 sub.u sin signo
mul multiplicación
 muls sin signo
 mulu con signo
 .d doble precisión en el destino.
div división
 divs sin signo
 divu con signo
 .d doble precisión en el destino.
cmp
```

$$\text{cmp} \begin{cases} \text{rD}, \text{rS1}, \text{rS2} \\ \text{rD}, \text{rS1}, \text{SIMM16} \end{cases}$$

|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |   |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|
| rD | nh | he | nb | be | hs | lo | ls | hi | ge | lt | le | gt | ne | eq | 0 | 0 |
|    | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1 | 0 |

(resto de bits a '0')

eq: 1 si y solo si rS1 = rS2

ne: 1 si y solo si rS1 ≠ rS2


gt: 1 si y solo si rS1 &gt; rS2 (con signo)

. . . . .

hi: 1 si y solo si rS1 &gt; rS2 (sin signo)

. . . . .

Bifurcaciones/saltos

| INST | Operandos |                                                                                                      |
|------|-----------|------------------------------------------------------------------------------------------------------|
| bb0  | B,rS1,D16 | PC ← PC+4*D16                                                                                        |
| bb1  | B,rS1,D16 | si bit B de rS1=  |
| br   | D26       | PC ← PC+4*D26                                                                                        |
| bsr  | D26       | r1←PC+4;PC←PC+4*D26                                                                                  |
| jmp  | (rS1)     | PC←rS1 (alineado)                                                                                    |
| jsr  | (rS1)     | r1←PC+4;PC←rS1 (alineado)                                                                            |

cmp r7, r2, r0;

bb1 2, r7, 20 == bb0 3, r7, 20

Todas las subrutinas las acabaremos con jmp (r1);

Transferencia (memoria)

| INST | Operandos       | Ext. / explicación |
|------|-----------------|--------------------|
| ld   | rD, rS1, SIMM16 | b,bu               |
|      | rD, rS1, rS2    | h,hu<br>d          |
| st   | rD, rS1, SIMM16 | b,h,d              |
|      | rD, rS1, rS2    |                    |
| ldcr | rD              | rD ← PSR           |
| stcr | rS1             | PSR ← rS1          |
| xmem | rD, rS1, rS2    | rD ↔ MEM(rS1+rS2)  |

ld como el load del IEEE

ejemplo: ld r10, r5, 0 r10 = 0xB4A39281

.b byte

ejemplo: ld.b r10, r5, 0 ; r10 = 0xFFFFFFFF81

|     |                  |                  |                    |
|-----|------------------|------------------|--------------------|
| .h  | half (medio)     |                  |                    |
|     | ejemplo:         | ld.h r10, r5, 0  | ; r10 = 0x00000081 |
| .d  | double (palabra) |                  |                    |
|     | ejemplo:         | ld.d r10, r5, 0  | ; r10 = 0xFFFF9281 |
| .bu | sin signo        |                  |                    |
|     | ejemplo:         | ld.bu r10, r5, 0 | ; r10 = 0xFFFF9281 |
| .hu | sin signo        |                  |                    |
|     | ejemplo:         | ld.hu r10, r5, 0 | ; r10 = 0x00009281 |

Campos de bit

| INST                             | Operandos                     |
|----------------------------------|-------------------------------|
| clr<br>set<br>ext<br>extu<br>mak | rD, rS1, W5<O5>               |
| rot                              | rD, rS1, <O5><br>rD, rS1, rS2 |

Instrucciones de coma flotante

| INST                                         | Operandos    | explicación                          |
|----------------------------------------------|--------------|--------------------------------------|
| fadd.xxx<br>fsub.xxx<br>fmul.xxx<br>fdiv.xxx | rD, rS1, rS2 | x=s $\bar{x}$ =d<br>x=d $\bar{x}$ =s |
| fcvt.x $\bar{x}$<br>flt.xs<br>int.sx         | rD, rS2      |                                      |
| fcmp.sxx                                     | rD, rS1, rS2 |                                      |

Ensamblador/Cargador

Bajar tanto el simulador (mc88110) como el compilador/ensamblador (88110e)

- Ensamblador: Programa que se encarga de “traducir” un programa escrito en lenguaje ensamblador a lenguaje máquina.  
etiqueta: instruccion\_ensamblador ; Comentarios
- etiqueta: es la dirección donde esta almacenada esa instrucción.
- Instrucción\_ensamblador: Puede ser una instrucción-máquina o una pseudoinstrucción.
- Pseudoinstrucción:
  - Instrucción para el programa ensamblador.
  - No se traduce en una instrucción en memoria, es decir, no se almacena en memoria.
  - Indica al ensamblador cómo debe generar el código-máquina.

Pseudoinstrucciones

- Org n: Indica que el código que le sigue se almacene en la posición de memoria n.
- Res n: Indica que se reserven n bytes en memoria. N debe estar alineado a palabra

- (múltiplo de 4). Los datos reservados no estan inicializados a nada.
- Data a, b, c, ....: Inicializa las posiciones de memoria con los valores a, b y c.  
ejemplo:  
data 7, "abc", -15 --> [[7], [a, b, c,  $\emptyset$ ], [-15]]
- Data "texto": Inicializa las posiciones de memoria con la cadena de bytes texto. Asegura que la siguiente palabra en memoria está alineada (véase ejemplo "data, más abajo").
- Low (etiqueta o inmediato): Devuelve los 16 bits menos significativos de la dirección asociada a la etiqueta o dato inmediato.
- High(etiqueta o inmediato): Devuelve los 16 bits más significativos de la dirección asociada a la etiqueta o dato inmediato.

### Macroinstrucciones ("macros")

- Conjunto de sentencias a las que se le asigna un nombre y se les pasa un conjunto de argumentos.
- Cuando aparece la invocación de la macro se sustituye en fase de ensamblado la macro por el conjunto de sentencias declarado en la macro cambiando los parámetros declarados por los que se pasan en la invocación.  

```

Nombre_de_macro: MACRO(arg1, arg2, ..., argn)
 Conjunto de instrucciones
 Que componen la macro
ENDMACRO

swap: MACRO(ra,rb)
 or r1,ra,ra
 or ra,rb,rb
 or rb,r1,r1
ENDMACRO

```
- Una macro debe haberse definido previamente.
- Se permiten macros anidadas.
- No se permite la definición de etiquetas dentro de una macro.
- Se utilizan para encapsular pequeños fragmentos de código para los que no merece la pena construir una subrutina.

tips

### Ejemplos

#### Ejemplo

```

org 100;
DATOS: res 200
OTROS: data 10, 20, 30
TXT: data "abcd\0" ; se guarda en ascii en memoria y Little
Endian
 ; abcd -> 61626364

```



|     |     |     |     |     |     |          |       |
|-----|-----|-----|-----|-----|-----|----------|-------|
| 100 | ... | 296 | 300 | 304 | 308 | 312      | 316   |
| ??  | ??  | ??  | 10  | 20  | 30  | 61626364 | 0???? |

Instrucciones parecidas

```
LD .R1, #OTROS; R1 <- 300
 or r1, r0, low(OTROS)
 or.u r1, r1, high(OTROS)

LD .R1, /OTROS; R1 <- (300) ; R1 <- 10
 or r1, r0, low(OTROS)
 or.u r1, r1, high(OTROS)
 ld r1, r1, r0
```

Ejemplo "data"

```
INI: ld r3, r0, 400 ; "data.ens"
 or r2, r0, low(numeros)
 or.u r2, r2, high(numeros)
 ; Las dos últimas IEEE: LD .R2, #NUMEROS
 stop
 org 400
 data 0x01020304

 org 412
 res 4
 data "SS00"
 data "1234567890abcdefgh\n\\0\\t"
numeros: data 15, 0x7AF, -5
```

```

practica@avellano% 88110e -o data.bin data.ens
88110.ens-INFO: Compilando data.ens ...
88110.ens-INFO: Compiladas 12 lineas
88110.ens-INFO: GenerandoCodigo...
88110.ens-INFO: Programa generado correctamente
practica@avellano%
```

```

practica@avellano% mc88110 data.bin
PC=0 ld r03,r00,400 Tot. Inst: 0 ij Ciclo : 1
FL=1 FE=1 FC=0 FV=0 FR=0
R01 = 00000000 h R02 = 00000000 h R03 = 00000000 h R04 = 00000000 h
R05 = 00000000 h R06 = 00000000 h R07 = 00000000 h R08 = 00000000 h
R09 = 00000000 h R10 = 00000000 h R11 = 00000000 h R12 = 00000000 h
R13 = 00000000 h R14 = 00000000 h R15 = 00000000 h R16 = 00000000 h
R17 = 00000000 h R18 = 00000000 h R19 = 00000000 h R20 = 00000000 h
R21 = 00000000 h R22 = 00000000 h R23 = 00000000 h R24 = 00000000 h
R25 = 00000000 h R26 = 00000000 h R27 = 00000000 h R28 = 00000000 h
```

R29 = 00000000 h R30 = 00000000 h R31 = 00000000 h  
88110>

88110> e

Fin ejecución

PC=16 instrucción incorrecta Tot. Inst: 4 ; Ciclo : 62

FL=1 FE=1 FC=0 FV=0 FR=0

R01 = 00000000 h R02 = 000001BC h R03 = 01020304 h R04 = 00000000 h  
R05 = 00000000 h R06 = 00000000 h R07 = 00000000 h R08 = 00000000 h  
R09 = 00000000 h R10 = 00000000 h R11 = 00000000 h R12 = 00000000 h  
R13 = 00000000 h R14 = 00000000 h R15 = 00000000 h R16 = 00000000 h  
R17 = 00000000 h R18 = 00000000 h R19 = 00000000 h R20 = 00000000 h  
R21 = 00000000 h R22 = 00000000 h R23 = 00000000 h R24 = 00000000 h  
R25 = 00000000 h R26 = 00000000 h R27 = 00000000 h R28 = 00000000 h  
R29 = 00000000 h R30 = 00000000 h R31 = 00000000 h

88110> v 400

|     |          |          |          |          |
|-----|----------|----------|----------|----------|
| 400 | 04030201 | 00000000 | 00000000 | 00000000 |
| 416 | 53534F4F | 31323334 | 35363738 | 39306162 |
| 432 | 63646566 | 67680A00 | 09000000 | 0F000000 |
| 448 | AF070000 | FBFFFFFF | 00000000 | 00000000 |
| 464 | 00000000 | 00000000 | 00000000 | 00000000 |

88110>

## Ejercicio

Ejercicios y Problemas (Instrucciones y direccionamientos) ([inst\\_dir\\_12-13.pdf](#))

11

### Problema

Considere un computador de 32 bits y direccionamiento a nivel de byte en el que a pila crece hacia direcciones decrecientes y el puntero de pila apunta a la primera posición libre. Todas las instrucciones del computador ocupan una palabra y la dirección de retorno de la subrutina se almacena en la pila. Dado el programa que se muestra a continuación, conteste a las siguientes preguntas. Suponga para ello que el programa principal está almacenado a partir de la dirección 0 y la subrutina a partir de 500, que el SP tiene el valor 5000, y los datos almacenados en las direcciones 1000 a 1016 son 1, 2, 3, 4 y 5.

| Programa principal                                                 | Subrutina (500:)                                                                                         | Interacción 1                                      | Interacción 2      |
|--------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------|----------------------------------------------------|--------------------|
| AND .R1, #0<br>LD .R2, #1000<br>LD .R3, #5<br>PUSH .R2<br>PUSH .R3 | LD .R1, #8 [.SP]<br>LD .R2, #12 [.SP]<br>LD .R3, #16 [.SP]<br>ADD .R1, [.R3++]<br>SUB .R2, #1<br>BNZ -12 | R1 ← 0<br>R2 ← 5<br>R3 ← 100<br>R1 ← 0 ; R3 ← 1004 | R1 ← 3 ; R3 ← 1008 |

|                                                           |                        |        |  |
|-----------------------------------------------------------|------------------------|--------|--|
| PUSH .R1<br>LD .R4, #500<br>CALL [.R4]<br>POP .R1<br>HALT | ST .R1, #8[.SP]<br>RET | R2 ← 4 |  |
|-----------------------------------------------------------|------------------------|--------|--|

1000 | 1                                      R1 ← 0 1 2 3 4 5 6 7 8 9 A  
 1004 | 2                                      R2 ← 1000 5 4 3 2 1 0  
 1008 | 3                                      R3 ← 5 1000 1004 1004 1008 100C 1010 1014  
 100C | 4                                      R4 ← 500  
 1010 | 5

1- Indique el contenido de la pila, registros (incluido el PC y el SP) y el biestable una vez ejecutada la instrucción CALL[.R4]

|           |
|-----------|
| DR = 0020 |
| R1 = 0    |
| R3 = 5    |
| RZ = 1000 |
| .         |

## Ejemplos (Drive)

### Ejemplo 1: Vectores

;Suma de un número variable de datos almacenados en posiciones consecutivas  
 ;de memoria a partir de la variable SUMDOS.  
 ;Cada dato ocupa una palabra y está representado en binario puro.  
 ;El numero de datos queda definido por la variable N y el resultado se almacena en la variable RESULT.  
 ;Si existe desbordamiento se devuelve -1 en el registro r30, caso contrario se devuelve 0 en dicho registro.

;Se definen las siguientes macros:

LEA: MACRO(ra,eti)                                      ;en el IEEE sería: LD    .R2, #eti  
       or        ra,r0,low(eti)

```

 or.u ra,ra,high(eti)
ENDMACRO
;Carga el registro "ra" con la dirección efectiva definida por la etiqueta
"eti".

```

```

DBNZ: MACRO(ra,eti)
 sub ra,ra,1
 cmp r4,ra,0
 bb0 2,r4,eti
ENDMACRO
;Decrementa el registro "ra" y salta a la etiqueta "eti" si no es cero el
resultado utilizando "r4" como registro auxiliar.

```

```

LOAD: MACRO(ra,eti) ;en el IEEE sería: LD .R2, /eti
 LEA (ra, eti)
 ld ra,ra,0

```

```

ENDMACRO
;Carga en el registro "ra" el contenido de la etiqueta "eti"

```

```

;Definición de datos utilizados por el programa

```

```

 org 1000
N: data 5
SUMDOS: data 0x80000000,24,0x2fffffff,0,3
RESULT: data 0

```

```

;
; Programa ensamblador
;

```

```

; registros utilizados:
; r0 siempre a cero
; r1 dirección de retorno de una subrutina
; r2 REUTILIZA. cmp.
; r3 resultado
; r5 número de elementos, utilizamos de contador
; r20 REUTILIZA.
; ...

```

```

 LEA (r20, SUMDOS)
LOAD (r5, N)
bucle: or r3, r0, r0 ; ini. r3 a 0
 cmp r2, r5, r0 ; cont.
 bb1 eq, r2, fin ; r2 libre
 ld r2, r20, r0 ; guardamos el dato del vector
 add r3, r3, r2 ; sumamos el dato al resultado
 subu r5, r5, 1 ; nº elem. quedan por sumar - 1

```

```

 addu r20, r20, 4 ; avanzamos el puntero al sig. elem.
 br bucle
fin: LEA (r20, RESULT) ; dir. para dejar el resultado
 st r3, r20, r0 ; resultado a la dir. mem.
 stop ; el simulador deja de ejecutar

```

; NOTA: ponemos la comparación antes del algoritmo para que valga para  
; cuando el vector es de tamaño cero.

```

;
;Tras la ejecución del programa a partir de la dirección de memoria 1000
;deben aparecer los valores:
; 992 05000000 00000080
;1008 18000000 FFFFFFF2F 00000000 03000000
;1024 1A0000B0 00000000 00000000 00000000

```

### Ejemplo 2: Matrices

```

;Una matriz de M filas y N columnas está almacenada por filas a partir de
;la variable MATRIZ. Cada elemento ocupa 1 palabra y está representado en
;binario
;puro. Sumar los elementos de cada columna dejando el resultado como un
;vector de doble precisión a partir de la variable VSUMA.
;Este vector está almacenado según el ordenamiento little-endian.

```

```

LEA: MACRO (ra, eti)
 or ra, r0, low(eti)
 or.u ra, ra, high(eti)

```

ENDMACRO

```

;Carga el registro ra con la dirección efectiva definida por la etiqueta
;"eti".

```

```

LOAD: MACRO (ra, eti)
 LEA (ra, eti)
 ld ra, ra, r0

```

ENDMACRO

```

;Cargará con el contenido de memoria que indica la etiqueta "eti".

```

```

DBNZ: MACRO (ra,eti)
 sub ra,ra,1
 cmp r5,ra,0
 bb0 2,r5,eti; 2->eq |

```

ENDMACRO

```

;Decrementa el registro ra y salta a la etiqueta "eti" si no es cero el
;resultado utilizando r5 como registro auxiliar.

```

```

;Datos del problema.

```

```

 org 1008
FILAS: data 5, 0
COLUMNAS: data 4, 0
MATRIZ: data 0xc000000b, 0xc000000a, 0xc0000009, 0xc0000008
 data 0xc0000007, 0xc0000006, 0xc0000005, 0xc0000004
 data 0xc0000003, 0xc0000002, 0xc0000001, 0xc0000000
 data 0xe000000b, 0xe000000a, 0xe0000009, 0xe0000008
 data 0xe0000007, 0xe0000006, 0xe0000005, 0xe0000004
VSUMA: res 32

```

```

; Programa ensamblador

```

```

; registros utilizados.
; r0 siempre a cero
; r1 dir retorno subrutina
; r2 el número de filas de la matriz
; r3 el número de columnas de la matriz
;

```

```

 LEA (r20, MATRIZ)
 LOAD (r2, FILAS)
 LOAD (r3, COLUMNAS)
 LEA (r22, VSUMA)
 mulu r4, r3, 3 ; desplazamiento
 or r21, r20, r20 ;
 or r5, r2, r2
bucle_fila: or r6, r0, r0 ; inicializamos a 0 r6, acumulador
bucle_col: cmp r7, r2, r0
 bb1 eq, r7, sig_col
 ld r7, r20, r2
 subu r2, r2, 1
 addu r20, r20, r4
 br bucle_col
sig_col: st r6, r22, r0 ; guardamos el resultado en memoria
 addu r22, r22, r4 ; avanzamos a la siguiente dirección
 ; para guardar el siguiente resultado
 or r2, r5, r5 ; reseteamos el valor de las columnas
 addu r21, r21, 4 ; apuntamos al 1º elem de sig. columna
 or r20, r21, r21 ; copiamos el puntero de la sig.
 column
 subu r3, r3, 1
 cmp r7, r2, r0
 bb1 ne, r7, bucle_fila
 stop

```

;Resultado del ejemplo:

```
; 1008 05000000 00000000 04000000 00000000
; 1024 0B0000C0 0A0000C0 090000C0 080000C0
; 1040 070000C0 060000C0 050000C0 040000C0
; 1056 030000C0 020000C0 010000C0 000000C0
; 1072 0B0000E0 0A0000E0 090000E0 080000E0
; 1088 070000E0 060000E0 050000E0 040000E0
; 1104 27000000 04000000 22000000 04000000
; 1120 1D000000 04000000 18000000 04000000
```

### Ejemplo 3: Lista no ordenada y compacta

;Insercion de un elemento en una lista no ordenada.

;La lista comienza a partir de la direccion definida por la etiqueta NOMBRE ;y cada elemento ocupa 1 palabra, estando almacenados en posiciones consecutivas de memoria. La longitud de la lista y el elemento a insertar vienen definidos por las variables LONG y NUEVO.

;Si el nuevo no se encuentra en la lista, se inserta al final de la misma y ;se incrementa la longitud. Si el nuevo se encuentra en la lista, no se ;hace nada.

```
org 1000
NOMBRE: data 6,5,7,4,80,-1,0,-3,101,10
org 2000
LONG: data 10
NUEVO: data 33
```

; macros

```
LEA: MACRO (ra, eti)
 or ra, r0, low(eti)
 or.u ra, ra, high(eti)
```

ENDMACRO

;Carga el registro ra con la dirección efectiva definida por la etiqueta ;“eti”.

```
LOAD: MACRO (ra, eti)
 LEA (ra, eti)
 ld ra, ra, r0
```

ENDMACRO

;Cargará con el contenido de memoria que indica la etiqueta “eti”.

```
DBNZ: MACRO (ra,eti)
 sub ra,ra,1
 cmp r5,ra,0
 bb0 2,r5,eti; 2->eq |
```

ENDMACRO

;Decrementa el registro ra y salta a la etiqueta “eti” si no es cero el

resultado utilizando r5 como registro auxiliar.

```
; registros utilizados
; ...
```

```
; codigo
 @TODO:
```

;Tras la ejecucion del programa quedaran en memoria los siguientes valores:

|        |          |            |          |          |
|--------|----------|------------|----------|----------|
| ; 2000 | 0B000000 | 21000000   | 00000000 | 00000000 |
| ; 992  |          |            | 06000000 | 05000000 |
| ; 1008 | 07000000 | 04000000   | 50000000 | FFFFFFFF |
| ; 1024 | 00000000 | FDFFFFFFFF | 65000000 | 0A000000 |
| ; 1040 | 21000000 |            |          |          |



Ejemplo 4: Lista ordenada y compacta

```
;Insercion de un elemento en una lista ordenada.
;La lista esta almacenada a partir de la direccion definida por LISTOR y
;esta constituida por elementos de 1 palabra de longitud expresados en
;binario puro.
;Estan ordenados de menor a mayor y estan almacenados en posiciones de
;memoria consecutivas. El tamaño de la lista esta almacenado en la
;direccion
;LONLIS y el elemento a insertar esta almacenado en la variable NUEVO.
;La lista no almacena elementos repetidos.
```

```
 org 1000
LISTOR: data 1,3,7,45,56,67,101,200
 org 2000
LONLIS: data 8
NUEVO: data 90
```

```
; macros
```

```
LEA: MACRO (ra, eti)
 or ra, r0, low(eti)
 or.u ra, ra, high(eti)
```

```
ENDMACRO
```

```
;Carga el registro ra con la dirección efectiva definida por la etiqueta
;"eti".
```

```
LOAD: MACRO (ra, eti)
 LEA (ra, eti)
 ld ra, ra, r0
```

```
ENDMACRO
```

```
;Cargará con el contenido de memoria que indica la etiqueta "eti".
```

```
DBNZ: MACRO (ra,eti)
 sub ra,ra,1
 cmp r5,ra,0
 bb0 2,r5,eti; 2->eq |
```

```
ENDMACRO
```

```
;Decrementa el registro ra y salta a la etiqueta "eti" si no es cero el
resultado utilizando r5 como registro auxiliar.
```

```
; registros usados
```

```
; ...
```

```
; code
```

```
 LEA (r20, LISTOR)
 LOAD (r3, LONGLIST)
 LOAD (r4, NUEVO)
```

```

 mulu r5, r3, r4
 or r21, r20, r0
 addu r21, r21, r5 ; 1º puntero al final
 or r22, r21, r0 ; 2º puntero al final
 subu r21, r21, 4 ; 1º puntero al anterior elemento
 or r5, r3, r0

 ; buscamos posicion a insertar
bucle: ld r2, r20, r0
 cmp r15, r2, r4
 bb1 eq, r15, fin ; si ya esta en la lista, no se
inserta
 bb1 gt, r15, desplazar; si hemos pasado, vamos a desplazar
 subu r5, r5, 1
 cmp r15, r5, r0 ; miramos si el contador es cero
 bb1 eq, r15, insertar ; final de lista se inserta al final
desplazar: ld r2, r21, r0
 cmp r15, r2, r4 ; también por dir. cmp r15, r22, r20
 bb1 lt, r15, insertar
 st r2, r22, r0
 subu r22, r22, 4
 subu r21, r21, 4
 br desplazar
insertar: st r4, r22, r0
fin: stop

```

```

;Tras la ejecucion del programa los contenidos de memoria son:
; 992 01000000 03000000
;
; 1008 07000000 2D000000 38000000 43000000
; 1024 5A000000 65000000 C8000000 00000000
;
; 2000 09000000 5A000000

```

Ejemplo 5: Lista encadenada

;Insercion de un elemento en una lista con punteros.  
 ;Cada elemento de la lista consta de dos palabras consecutivas en memoria,  
 la primera ;contiene la direccion del siguiente elemento de la la lista y  
 ;la segunda el valor numerico del ;elemento, que esta representado en  
 ;binario puro.  
 ;La lista esta ordenada de menor a mayor.  
 ;La variable CABECERA contiene la direccion del primer elemento de la  
 lista ;y la lista termina ;cuando el campo de direccion de un elemento  
 contiene ;el valor 0x00000000. Por lo tanto, si ;la variable CABECERA  
 contiene el ;valor cero es que la lista esta vaca.  
 ;La variable NUEVO contiene la direccion del elemento nuevo a insertar en  
 ;la lista.

;DATOS

```

 org 1000
CABECERA: data 2000
NUEVO: data 2200

```

;EJEMPLO PARA COMPROBACION

```

 org 2000
 data 2024,2
 org 2024
 data 2048,5
 org 2048
 data 2072,10
 org 2072
 data 2096,13
 org 2096
 data 0,15
 org 2200
 data 2112,7

```

; macros

```

LEA: MACRO (ra, eti)
 or ra, r0, low(eti)
 or.u ra, ra, high(eti)

```

ENDMACRO

;Carga el registro ra con la dirección efectiva definida por la etiqueta  
 ;“eti”.

```

LOAD: MACRO (ra, eti)
 LEA (ra, eti)
 ld ra, ra, r0

```

ENDMACRO

;Cargará con el contenido de memoria que indica la etiqueta “eti”.

```

DBNZ: MACRO (ra,eti)
 sub ra,ra,1
 cmp r5,ra,0
 bb0 2,r5,eti; 2->eq |

```

```
ENDMACRO
```

;Decrementa el registro ra y salta a la etiqueta “eti” si no es cero el resultado utilizando r5 como registro auxiliar.

```

; registros usados
; ...

```

```
; code
```

;Tras la ejecucion del programa el contenido de memoria queda:

```

; 2000 E8070000 02000000 00000000 00000000
; (2024) (a1)
; 2016 00000000 00000000 98080000 05000000
; (2200) (a2)
; 2032 00000000 00000000 00000000 00000000
; 2048 18080000 0A000000 00000000 00000000
; (2072) (a4)
; 2064 00000000 00000000 30080000 0D000000
; (2096) (a5)
; 2080 00000000 00000000 00000000 00000000
; 2096 00000000 0F000000 00000000 00000000
; (fin) (a6)
; 2112 00000000 00000000 00000000 00000000
; 2128 00000000 00000000 00000000 00000000
; 2144 00000000 00000000 00000000 00000000
; 2160 00000000 00000000 00000000 00000000
; 2176 00000000 00000000 00000000 00000000
; 2192 00000000 00000000 00080000 07000000
; (2048) (a3)

```

## Subrutinas

Código encapsulado con una especificación muy bien definida que se puede utilizar desde varios puntos del programa. Una vez realizado su código el programa volverá al lugar desde donde se le llamó.

#### PXXXXX a la subrutina:

- Antes de llamar a la subrutina se almacenan en la pila para que los pueda tocar la subrutina. Luego recuperas los registros que se encuentran en la pila.
- La otra opción es que la subrutina guarde y saque las cosas, solo se da en la de interrupción y las de manera asíncronas.

#### Tipos de variables que utiliza una subrutinas

- Variables globales: se crean al inicio y se destruyen al final, ámbito global. Para nosotros.
- Variables locales: serán las que tienen etiquetas o por su dirección absoluta en memoria sólo tienen validez en una subrutina. Su ámbito es solo en esa llamada de subrutina.

#### Paso de parámetro a la subrutina:

- Por valor:
- Por dirección: cuando va a ser un parámetro de salida. cuando es de entrada-salida. Cuando ocupa más que no merece la pena hacer una copia de tanto para solo modificar un solo dato.

#### Donde pasamos los parámetros:

- En registros: un acuerdo de donde se pasan los parámetros. Limitado al número de registros generales disponibles y limitados a palabra.
- En variables globales: Acuerdo en dos posiciones de memoria fijas. Problemas de reentrancia, es decir, una subrutina se dice que es reentrante si puede ser llamada antes de que finalice su propia ejecución, esto se suele dar cuando hay asincronismo.
- En la pila: Una estructura de datos de tipo LIFO. Se necesita tener instrucciones para meter los parámetros en la pila y para sacarlos. Para ir a la subrutina, se utilizan las llamadas. Los datos de salida no van en la pila.

#### En el 88110

- No tiene puntero de pila, se utilizará el r30.
- La dirección de retorno de la subrutina es en el r1.
- No tiene ni PUSH ni iPOP, por tanto hay que crear dos macros con esos nombres.
- Con subrutinas anidadas hay que salvaguardar el r1 en la pila al principio de la subrutina (SIEMPRE), ya que lo va a sobrescribir la subrutina siguiente. Se hace siempre que no se esté en una subrutina hoja, las que no llaman a ninguna otra.
- El RET del IEEE, aquí es un jmp(r1).

¿?Registro de activación de la subrutina

#### Marco de pila:

- Todos los datos privados de la subrutina, parámetros, direcciones de retorno y locales.
- Cada subrutina tiene su propio marco de pila.

### Puntero de marco de pila

- Se decide un registro que apunta a una posición conocida del marco de pila.
- Será el registro r31.
- Crear un espacio necesario para las variables locales y las inicializaciones de esas variables locales.

Todos los elementos del vector tienen que estar a media palabra.

...  
...  
...

### Ejemplo

Factorial en el 88110

```
fact: PUSH (r1)
 ; devolvemos el valor de retorno r29

 ld r2, r30, 4
 cmp r3, r2, r0
 bb1 eq, r3, fin
 subu r2, r2, 1
 bsr fact ; (n-1)!
 addu r30, r30, r30
 ld r2, r30, 4 ; n
 mulu r29, r29, r2 ; n!

salir: POP (r1)
 jmp (r1)
fin: addu r29, r0, 1
 br salir;
```

El registro r30, será el puntero de pila. Las instrucciones Push y Pop no están disponibles,

La subrutina va a modificar r3, y r7 ¿?NO SE POR QUE

En los programas de alto nivel, hay dos **tipos de variables**:

Variable local en subrutina: Es una variable que sólo existe mientras que se está ejecutando la subrutina. Cuando se deja de ejecutar la variable local se elimina.

Variable global en subrutina: La variable global es visible para todos, programa llamante principal y subrutina.

También hay dos **tipos de parámetros**:

Los parámetros se pasan por dirección, se da la dirección de memoria del parámetro.

Ejemplo: Suma  $r20 = r56 + r30 = 9$

$r56 = 5$

$r30 = 4$

Si los parámetros se se pasa por valor, directamente se pasan los datos con los que se va a operar.

Mismo ejemplo anterior:

$r20 = 5 + 4 = 9$

Organización en memoria:

Al no haber puntero de pila, hay que meterlo en direcciones de memoria.

Todas las subrutinas comenzarán con PUSH (r1)

Marco de Pila.

Conjunto de datos privados a una subrutina que incluye parámetros, dirección de retorno y variables locales.

Ejercicio 6. Recorido de lista encadenada de caracteres con objeto de determinar su longitud y el número total de caracteres de la lista.

La lista esta almacenada

LOAD (r20, LISTAENC)

LOAD (r21, LISTALONG)

LEA (r22, NELEM)

```

 LEA (r23, NCAR)
 ;init cont
 or r11, r0, r0
 or r12, r0, r0
 ;Si cadena vacía FIN (LISTAENC)
BUC_EL: cmp r2, r20, r0
 bb1 eq, r2, FIN

 :trat cadena
 or r10, r0, r0 ;cont.cad=0
BUC_CAD: Ld.bu r1, r20, r0 ;r1 ← car
 addu r10, r10, 1
 addu r20, r20, 1
 cmp r2, r1, r0 ;si car != 0 a BUC_CAD
 bb0 eq, r2, BUC_CAD
 ;si cadena vacía Fin (LISTALONG)
 cmp r2, r21, r0
 bb1 eq, r2, FIN
 st r10, r21, r0
 ;incrementa contadores totales
 addu r11, r11, 1
 addu r12, r12, r10
 ;avanzar puntero r20 a sig. dir. alineada
 mask r3, r20, 3 ; || and r3, r20, 3
 cmp r2, r3, r0
 bb1 eq, r2, ALINEADA
 and r20, r20, 0xFFFC
 addu r20, r20, 4
 addu r20, r20, 3
 and r20, r20, 0xFFFC
ALINEADA: Ld r20, r20, r0 ; avanzar punt. r20
 Ld r21, r21, 4 ; avanzar punt. r21
 br BUC_EL
FIN: st r11, r22, 0
 st r12, r23, 0
 stop

 ; r11(cont.nelem) → M(NELEM)
 ; r12(cont.cartot) → M(NCAR)

```



```

SUBROUTINA (int p1, int p2, int p3) {
 int vl - a = 0;
 char vl - b = 0;
 char vl - c = 45;
 /*fragmento 1*/

 /*fragmento 2*/
 funcion(0, p2, vl-a);
 /*fragmento 3*/

 /*FIN*/
}

```

*Primer fragmento:* Creación del marco de pila e inicialización de variables locales

```

SUBROUTINA: PUSH(r1)
 PUSH(r31)
 or r31, r30, 0
 sabu r30, r30, 8 ;reservar esp. v.l
 ;inic. var. locales
 st r0, r31, -4 ;vl - a = 0
 st.b r0, r31, -5 ;vl - b = 0
 or r2, r0, 45
 st.b r2, r31, -6 ;vl - c = 45

```

```

b) PUSH (r9)
 PUSH (r10)
 ld r2, r31, -4
 PUSH (r2)
 ld r2, r31, 12
 PUSH (r2)
 PUSH (r0)
 bsr FUNCION

```

```

c) or r30, r31, r0
 POP (r31)
 POP (r1)
 jmp (r1)

```

### **Problema 8:**

```

a)
CEROS: PUSH (r1)

```

```

ld r3, r30, 4 ;r3 ← n
ld r20, r30, 8 ;r20 ← puntero a vector
or r29, r0, r0 ;cta = 0
BUC: ld r5, r20, 0 ;r5 ← vector(i)
 cmp r2, r5, r0
 bb0 eq, r2, sigue
 addu r29, r29, 1

```

Sigue:

*Lunes 14 de octubre de 2013*

b)

```

DIAGONAL: PUSH (r1)
 PUSH (r31)
 or r31, r30, r0 ; cta = 0
 ; reserva espacio v.l.(vector)=m·4
 ld r4, r30, 8 ; r4 ← m
 mulu r3, r4, 4 ; r3 ← m·4
 sub r30, r30, r3 ;
 ; inic. puntero matriz y vector
 ld r20, r31, 12 ; r20 <- matriz
 or r21, r30, r0 ; r21 <- vector
 addu r3, r3, 4 ; r3 <- (m+1)·4
BUC: ld r18, r20, 0
 st r18, r21, 0
 ; avanza punteros
 addu r20, r0, r3 ; avanza r20 (m+1)·4
 add r21, r21, 4
 DBNZ (r4, buc)
DIAGONAL: PUSH (r30) ; Se puede sustituir por:
 ; or r21, r30, 0 ; r21 <- vector
 ; PUSH(r21)
 ; r4 <- m
 ld r4, r31, 8
 PUSH (r4)
 bsr CEROS
 addu r30, r30, 8 ; sacar par de la pila
 subu r5, r4, r29 ; r5 = n° elementos != 0
 ld r6, r31, 16 ; r6 <- RESULT
 st r5, r6, 0 ; r5 = n° elementos != 0 -> M(RESULT)

```

```
or r30, r31, r0
POP (r31)
POP (r1)
jmp (r1)
```

## TEMA 3 - PROCESADOR

### Objetivos

Fundamentos de los computadores, Pedro de Miguel

### Introducción

Estudiaremos la unidad de control y los caminos

Recordemos el tema 1, el de introducción, la parte del computador y la CPU y sus componentes.

Funciones básicas de la CPU (manda la UC las señales): Ejecutar instrucciones

- Leer la instrucción en memoria.
- Decodificar la instrucción
- Ejecución: como máximo.
  - Búsqueda de operandos
    - Cálculo de operando
    - Leer operando en memoria
  - Operación
  - Almacenar resultado
- Preparar la siguiente instrucción

### Unidad de control

La Unidad de Control, manda las señales para que todo se ejecute correctamente.

Funciones de la UC:

1. Ejecuta instrucciones (función básica)
  - a. Lectura, decodificación, e interpretación de las instrucciones
  - b. Generación de órdenes para la ejecución
  - c. Secuenciamiento de las instrucciones (decidir cuál es la siguiente a y.gt)
2. Resuelve situaciones anómalas (desbordamiento, operación no válida, error de paridad, etc).
3. Controla la comunicación con periféricos

Entradas de la UC

- IR (Registro de Instrucción)
- Reloj
- SR (Registro de estado)
- Señales: E/S INT, busrq

Salidas de la UC

- Señales de control

## Reloj

Reloj: tren de pulsos caracterizado por su periodo

- señales de control: siempre sincronizadas con el reloj
  - Define el tiempo de cada operación.
  - Memoria: tiempo de lectura y escritura
- ALU: tiempo de operación, es el componente más lento de la CPU.
- CAMINO CRÍTICO: camino de máximo retardo entre un origen y un destino. Depende de los dispositivos que tengan que atravesar las señales.

Se pueden:

- nivel
- flanco de subida
- flanco de bajada

La duración del ciclo de reloj, se fija a la cantidad de tiempo que tarda en hacer una operación básica dentro de la CPU. es mucho mayor a la que pueda hacer de un registro a otro. Pero tendremos que darle igualmente un ciclo de reloj en el cronograma.

Ejemplo:

si  $f = 100 \text{ MHz}$   $\rightarrow T = 10 \text{ ns}$  y conviene que una operación aritmética se realice en un tiempo algo menor que 10 ns

¿y la Memoria? se le dará el cociente entero superior de dividir lo que tarda la memoria entre el T.

-----  
xxx  
-----

Ejemplo:

Cuantos ciclos de bus tiene ADD [3], [4]

fetch - 1 ciclo de bus, por que hay que leer la instrucción

$M(r3)+M(r4) \rightarrow M(r3)$  - hay 3 ciclos al leer los registros

Los buses tiene tantos cables como bits tiene una palabra si son buses paralelo (normalmente).

## Problemas

Hoja de uc\_problemas13-14\_s

[Problema 1](#)

@TODO: pasarlo de las hojas de FIWIKI.

Problema 3

- a) No, ya que las instrucciones aritmético lógicas son las que lo modifican, y luego en los saltos condicionales se necesita leer el registro de estado o cuando haya que resolver cosas excepcionales pero para la de comparación no.
- b) Si es de propósito específico, tiene la instrucción, no tiene direcciones y menos la de la siguiente instrucción, eso es del PC.
- c) a
- d) a

Problema 4

Tacc = 24 ut

Top = 16 ut -> tk 16ut

mem tiene 2 ciclos =  $24 / 16 = 1,5 \rightarrow 2$

ADD #5[.3++], /dir 2 palabras  $M(5+R3) \leftarrow M(5+R3) + M(\text{dir}); R3 \leftarrow R3 + 1$

a)

NOTA: como no tenemos el esquema, suponemos uno.

suponemos para aclararnos:  $/1000 = 5$  ;  $/105 = 7$  ;  $R3 = 100$

Suponemos máquina a nivel de palabra, para el PC+1 y no +4 igual que en l ++ será +1

Fetch: suponemos que ya está hecho el fetch y RI tiene la primera palabra de la instrucción

Deco:

Ejec:

PC  $\rightarrow$  AR 1 ciclo

M(AR)  $\rightarrow$  DR 2 ciclos ; DR=1000

PC+1  $\rightarrow$  PC 1 ciclo

DR  $\rightarrow$  AR 1 ciclo

M(AR)  $\rightarrow$  DR 2 ciclos, ; DR=5

DR  $\rightarrow$  TEMP 1 ciclo, tenemos el 2 operando, dir

RI(despl) + BR(R3)  $\rightarrow$  AR 1 ciclo, la dir del primer operando con su despl

M(AR)  $\rightarrow$  DR 2 ciclos, ya tenemos dato 1º operando con su despl

DR + TEMP  $\rightarrow$  ~~DR~~ TEMP ; RE

1 ciclo (conflitto en el bus por eso en el temp2) ; Actualiza SR o RE (estado)

TEMPz $\leftarrow$  DR 1 ciclo

DR  $\rightarrow$  M(AR) 2 ciclos

R3 + 1  $\rightarrow$  R3 1 ciclo

Fetch: siguiente instrucción

PC → AR

M(AR) → DR

PC + 1 → PC

DR → RI ; Ir a C.O

Agrupamos

Fetch: suponemos que ya está hecho RI tiene la primera palabra de la instrucción

Deco:

Ejec:

PC → AR                      1 ciclo

M(AR) → DR ; PC+1 → PC    Suponemos a nivel de palabra                      2 ciclos

DR → AR                      1 ciclo

M(AR) → DR                      2 ciclos, ponemos la dir

DR → TEMP                      1 ciclo, tenemos el 2 operando, dir

RI(despl) + BR(R3) → AR                      1 ciclo, la dir del primer operando con su despl

M(AR) → DR , ya tenemos dato 1º operando con su despl ; R3 + 1 → R3    2 ciclos

DR + TEMP → TEMP2 ; RE    1 ciclo (conflitto en el bus por eso en el temp)

TEMP2 → DR                      1 ciclo

DR → M(AR) ; R3 + 1 → R3 ; PC → AR 2 ciclo

Fetch: siguiente instrucción

M(AR) → DR ; PC + 1 → PC    2 ciclos

DR → RI ; ir a C.O

b)

Cableada:

Dependerá de los solapamientos de las instrucciones

Teje = 12c + 5\*1c = 17c \* 16 ut/c = 272 ut

Teje = 10 + 4 (ciclo extra pro accesos a memoria) + 3+1 (fetch) = 18 ut

microprogramada

Tck = Tre + Tsec + 2\*TmicroPC + Tmc + 2\*Trc + Top + Tre + Tbr = 14 + 18 + 16 = 48 ut

Teje = 18 ciclos + Tck = 18c \* 48ut/c =

## Problema 8

Memoria asíncrona -> RDY (listo)

Tacc = 110 ut

a3: bucle M(AR) → DR ; si !RDY ir a a3

a)

Hay que nombrar cada flecha a los buses desde los registros, para poder acceder a ellos.  
Hay que nombrar las señales de cargas en los registro su que se activan por flanco.

b)

como es microprogramada:

TMPA tarda mas tiempo, solo se considera el mayor de los dos,

Top = Tr + Tmux + Tam + {Tbr, Tre} = 4 + 1 + 54 + 8 = 67

Tsec = Tre + Tmux + Tdec + Tmux + TmicroPC = 4 + 1 + 5 + 1 + 4 = 15 ut

Tmc = TmicroPC + 38 ut + Trc = 4 + 38 + 4 = 46 ut

Tck = Trc + Top + Tsec + Tmc = 4 + 67 + 15 + 46 = 132 ut

c)

dir Rel a RB

RB -> TMPB; RI(desp) -> TMPA

TMPA + TMPB -> TMPC

TMPC -> TMPB

TMPB -> AR

-----

M(AR) -> DR

### Problema 15

@TODO: mirar bien este ejercicio

Suponemos la misma máquina del problema 8 con memoria asíncrona

OR .R1, .R2, /dir ; 2 palabras ; R1 <- R2 v M(dir)

Suponemos dir = 1000, y en el 1000 hay un 5

El PC apunta a la segunda palabra

PC -> TMPB

TMPB .> AR ;

M(AR) -> DR ; TMPB+1 -> PC ; Si !RDY bucle aqui ; DR tiene el valor 1000

DR -> TMPC

TMPC -> TMPB

TMPB -> AR

M(AR) -> DR ; Si !RDY bucle aqui ; BR(R2) -> TMPB ; DR tiene el valor 5



DR v TMPB -> BR(R1) ; Act. RE

Para el fetch hay dos alternativas, o se solapan con las anteriores o saltas a una micro rutina (Ir al fetch)

fetch:

PC -> TMPB

TMPB .-> AR ; TMPB+1 -> PC

M(AR) -> DR ; Si !RDY bucle aqui ; DR tiene el valor 1000

DR -> TMPC; Ir a C.O

b)

Tacc = 110 ut

Teje =  $8+2 + 4+1 = 15c = 15*132ut/c = 1980 ut$

con la máquina del enunciado

gen4 (para el PC)

0+CY=1 -> TMP1

TMP1\*2 -> TMP1

TMP1\*2 -> TMP1; microret

fetch

PC -> AR; microCALL gen4

M(AR) -> DR ; PC+TMP1 -> PC

9

Se puede entender que existe una microoperación que se llama, “actualizar registro de estado”.

Algunas instrucciones afectan a los flags.

(Diapositivas, deltapaf de Pedro de Miguel)

Registros

Especiales

PC

PC+1 --> PC

y “3<□ □ >”

OP “y+1<□ □ >”

TD “□ □ □ ”

FP “\_”

SP

RI (no visible al programador)

Transparentes

RA NO VISIBLE AL PROGRAMADOR

AR registros de direcciones

DR registros de datos

Ejemplo

ADD R4, R3

PSEUDOCODIGO

fetch

“M(PC) --&gt; RI”

PC --&gt; AR

Microinstrucción 1 ciclos

M(AR) --&gt; DR

Microinstrucción 2 ciclos

DR --&gt; RI

Microinstrucción 1 ciclos

“PC+1 --&gt; PC”

decodificacion

“Llevar a R4 la suma de R4 y R3” R4+R3 --&gt; R4

“Modificar el RE”

SOLUCIÓN

|     |                 |                  |          |
|-----|-----------------|------------------|----------|
| i   | PC --> AR       | Microinstrucción | 1 ciclos |
| i+2 | M(AR) --> DR    | Microinstrucción | 2 ciclos |
| i+3 | PC+1 --> PC     | Microinstrucción | 1 ciclos |
| i+4 | DR --> RI       | Microinstrucción | 1 ciclos |
| i+5 | decodificacion  | Microinstrucción | 1 ciclos |
| i+6 | R4+R3 --> R4    | Microinstrucción | 1 ciclos |
| i+7 | Modificar el RE | Microinstrucción | 1 ciclos |

28 / SEP / 2013

Ejemplo:

|      |                                      |
|------|--------------------------------------|
| ADD  | R <sub>i</sub> , R <sub>j</sub>      |
| ADD  | R <sub>i</sub> , [++R <sub>j</sub> ] |
| LD   | R <sub>i</sub> , /dir. Memoria       |
| PUSH | R <sub>i</sub>                       |
| POP  | R <sub>i</sub>                       |
| BRZ  | /dir. Memoria                        |

---

29 / SEP / 2013

Hoja procesador 2 1213  
Ejercicio 1

Fases

Fetch (1ª palabra)  
Decodificación  
Fetch ( 2ª palabra)  
Ejecución:

“ $R1 \leftarrow R2 \text{ OR } M(\text{dir})$ ”

Actualizar RE

Decodificación de las fases en microoperaciones

Fetch (1ª palabra)

“ $M(\text{PC}) \rightarrow \text{IR}$ ”

PC  $\rightarrow$  AR

PC  $\rightarrow$  TMPB

TMPB  $\rightarrow$  AR

M(AR)  $\rightarrow$  IR

M(AR)  $\rightarrow$  DR

DR  $\rightarrow$  IR

“PC + 1  $\rightarrow$  PC”

PC  $\rightarrow$  TMPA(AR)

TMPA + 1  $\rightarrow$  PC

Decodificación

Fetch (2ª palabra)

“M(PC)  $\rightarrow$  IR”

PC  $\rightarrow$  AR

PC  $\rightarrow$  TMPB

TMPB  $\rightarrow$  AR

M(AR)  $\rightarrow$  IR

M(AR)  $\rightarrow$  DR

DR  $\rightarrow$  IR

“PC + 1  $\rightarrow$  PC”

PC  $\rightarrow$  TMPA(AR)

TMPA + 1  $\rightarrow$  PC

Ejecución:

“R1  $\leftarrow$  R2 OR M(dir)”

@TODO

Actualizar RE

### Secuencia de microoperaciones

i: # fetch 1ª palabra

PC  $\rightarrow$  TRPB

PC  $\rightarrow$  TRPA

i + 1:

TRPB  $\rightarrow$  AR

TRPA + 1  $\rightarrow$  PC

i + 2:  
M(AR) → DR                   # dura dos ciclos de reloj

i + 3:  
M(AR) → DR                   # dura dos ciclos de reloj

i + 4:  
DR → IR

i + 5:  
DECODIFICAR

i + 6: # fetch 2ª palabra  
PC → TRPB  
PC → TRPA

i + 7:  
TRPB → AR  
TRPA + 1 → PC

i + 8:  
M(AR) → DR                   # dura dos ciclos de reloj

i + 9:  
M(AR) → DR                   # dura dos ciclos de reloj

i + 10:  
DR → TMPC

i + 11:  
TMPC → TMPB

i + 12:  
TMPB → AR

i + 13:  
M(AR) → DR                   # dura dos ciclos de reloj  
R2 → TMPB

i + 14:  
M(AR) → DR                   # dura dos ciclos de reloj

i + 15:  
DR or TMPB → R1  
  
actualizar el RE

Hoja procesador 2 1213

Ejercicio 3

### Instrucción

LD .R3, /dir

### Fases

Fetch (1ª palabra)

Decodificación

Fetch ( 2ª palabra)

Ejecución:

“R3 ← M(dir)”

### Decodificación de las fases en microoperaciones

Fetch (1ª palabra)

PC → AR

M(AR) → DR # 2 ciclos

DR → IR

PC + 1 → PC

**30 / OCT/ 2013**

LD .R3, /dir(2p)

fetch

“M(PC) → R4”

PC → AR

M(AR) → DR

DR → RI

PC + 1 → PC

decodif

fetch (2ª pal)

“M(PC) → DR”

PC → AR

M(AR) → DR

ejecución

“M(DR) → R3”

DR → AR

"M(AR) → R3"

M(AR) → DR

DR → R3

PC + 1 → PC

04 / NOV / 2013

Ejercicio 4 del Juego de problemas 2

Juego 2. Plv1, Plv 3.

La máquina de Pedro.

Reposo:

Secuenciamiento

Código de secuencia | Condición

Microsalto si se cumple la condición:

Mdir

Micro PC + 1

Micr salto a código de operacion "CO"

Al hacer el fetch → Comienzo del microprograma de la instrucción.

-----

fetch

"M(PC) → R"

PC → R

M (AR) → DR (2 ciclos)

DR → AR

PC + 8 → PC

PC → T1

T1 + 8 → PC

Código de Operaciones

CALLNZ #despl [++.R2]

----- (¿fetch?)

"PC → M(SP)"

PC → DR

SP → AR

DR → M(AR) (2 ciclos)

"SP-8 → SP"

SP → T1



$T1 - 8 \rightarrow SP$   
 "SP-8  $\rightarrow$  SP"  
 $R2 \rightarrow T1$   
 $T2 + 8 \rightarrow R2$   
 $T2 + 8 \rightarrow T2$   
 "R2 + desplazamiento  $\rightarrow$  PC"  
 $IR(\text{desplazamiento}) \rightarrow T1$   
 $T1 + T2 \rightarrow T1$   
 microSalto a fetch 1

### Secuencia de microoperaciones

i:      # fetch  
         PC  $\rightarrow$  AR                  microOp. + 1  
         PC  $\rightarrow$  T1                  microOp. + 1  
 i + 1:  
         T1 + 8  $\rightarrow$  PC              microOp. + 1  
         M(AR)  $\rightarrow$  DR              microOp. + 1              # dura dos ciclos de reloj  
 i + 2:  
         M(AR)  $\rightarrow$  DR              microOp. + 1              # dura dos ciclos de reloj  
 i + 3:  
         DR  $\rightarrow$  IR                  microSalto a Cod.Op.

----- COMPLETAR -----

## TEMA 4 - REPRESENTACIÓN Y ARITMÉTICA

ANTONIO PEREZ, DESPACHO 4108 (LAB. ARQUITECTURA DE COMPUTADORES)

[aperez@fi.upm.es](mailto:aperez@fi.upm.es)

1. Introducción
  - 1.1 Representaciones alfanuméricas y numéricas
  - 1.2 Operador y estructura de la ALU
2. Representación en coma fija
  - 2.1 Binario sin signo
  - 2.2 Complemento a 2, complemento a 1 y signo magnitud
  - 2.3 Exceso a m
3. Representación en coma flotante
  - 3.1 Definición, rango y resolución
  - 3.2 Normalización y bit implícito
  - 3.3 Suma y resta
  - 3.4 Redondeo y bits de guarda
  - 3.5 Estandar IEEE 754
4. Otras operaciones.

Recomienda libro Omondi.

### Representación de la información

Al computador le llegan instrucciones y datos (definidos por símbolos e ideas) y produce resultados.

Condicionantes del computador: Está constituido por una serie de circuitos integrados, hace años la representación era mediante dos estados, un estado de ausencia de tensión (0 lógico) y otro de entrada de tensión (1 lógico).

Modos de representación:

- Representaciones alfanuméricas.
- Representaciones numéricas. IMPORTANTE
- Representaciones redundantes.
- Representaciones características.

### Representaciones alfanuméricas

Se usará la representación alfanumérica cuando se quieran representar los símbolos como

caracteres. Se utilizará la representación numérica cuando se quieran hacer cálculos con los datos.

Representan:

- 26 letras del alfabeto (Mayus y minus).
- 10 dígitos decimales.
- Caracteres especiales.
- Caracteres de control.

Características:

ASCII → American Standard Code for Information Interchange.

EBCDIC → Lo utilizo IBM durante muchos años. Su representación del lenguaje alfanumérico.

- Facilidad para comprobar un carácter numérico  
ASCII: desde H'30 hasta H'39
- Fácil equivalencia Mayúsculas y minúsculas  
ASCII: desde H'41 (A) hasta H'5A (Z)  
ASCII: desde H'61 (a) hasta H'7A (z)
- Fácil comprobación si es carácter de control  
ASCII: desde H'00 (NUL) hasta H'1F (US)  
ASCII: excepción H'7F (DEL)

Flujo de información dentro del computador. Imaginad que tenemos un computador, y tenemos una pantalla (Terminal X) con un teclado, imaginaos que tengo una calculadora. Quiero decirle "53 + 35 =" y que me conteste su respuesta. Voy al teclado, pulso las cosas y...

¿Qué es lo que fluye por esta línea de comunicación entre el computador y el teclado?  
El código que fluye es el código ASCII.

ASCII H'35 H'33 H'2B H'33 H'35 H'3D

BCD 5 3 + 3 5 =

Binario 110101 + 100011

Binario 1011000

BCD 88

BCD H'38 H'38

|                              |     | Carácter más significativo |     |       |   |   |   |   |     |
|------------------------------|-----|----------------------------|-----|-------|---|---|---|---|-----|
| Carácter menos significativo | HEX | 0                          | 1   | 2     | 3 | 4 | 5 | 6 | 7   |
|                              | 0   | NUL                        | DLE | Space | 0 | @ | P | ` | p   |
|                              | 1   | SOH                        | DC1 | !     | 1 | A | Q | a | q   |
|                              | 2   | STX                        | DC2 | "     | 2 | B | R | b | r   |
|                              | 3   | ETX                        | DC3 | #     | 3 | C | S | c | s   |
|                              | 4   | EOT                        | DC4 | \$    | 4 | D | T | d | t   |
|                              | 5   | ENQ                        | NAK | %     | 5 | E | U | e | u   |
|                              | 6   | ACK                        | SYN | &     | 6 | F | V | f | v   |
|                              | 7   | Bell                       | ETB | '     | 7 | G | W | g | w   |
|                              | 8   | BS                         | CAN | (     | 8 | H | X | h | x   |
|                              | 9   | HT                         | EM  | )     | 9 | I | Y | i | y   |
|                              | A   | LF                         | SUB | *     | : | J | Z | j | z   |
|                              | B   | VT                         | ESC | +     | ; | K | [ | k | {   |
|                              | C   | FF                         | FS  | ,     | < | L | \ | l |     |
|                              | D   | CR                         | GS  | -     | = | M | ] | m | }   |
|                              | E   | SO                         | RS  | .     | > | N | ^ | n | ~   |
|                              | F   | SI                         | US  | /     | ? | O | _ | o | DEL |

## Representaciones numéricas

### Limitaciones de una representación

¿Qué limitaciones tiene una representación?

Cuando estudiasteis hace años en el bachillerato, nos enseñaron que había una serie de números naturales, los enteros capaces de representar los números naturales más el cero en el computador?

- **NÚMERO FINITO DE NÚMEROS REPRESENTABLE:**  
Rango de representación: Intervalo entre el mayor y el menor número representables.
- **NÚMERO FINITO DE BITS PARA LA REPRESENTACIÓN:**  
Resolución: Diferencia entre los valores de un número representable y el inmediato siguiente.
- **OPERACIONES CON RESULTADOS NO REPRESENTABLES:**  
Desbordamiento: El resultado está fuera del rango de representación.

### SISTEMAS POSICIONALES CON BASE

$b = \text{base} = n^{\circ} \text{ natural} > 1$

$\text{Rep}(X) = (... x_2 x_1 x_0 x_{-1} x_{-2} ...) \text{ con } x_i \in \{b-1, b-2, \dots, 1, 0\}$

$$V(X) = \sum_{i=-\infty}^{\infty} x_i b^i = \sum_{i=0}^{\infty} x_i b^i + \sum_{i=1}^{\infty} x_{-i} b^{-i}$$

Nosotros, por comodidad, solemos utilizar la base 16, porque dígitos empaquetados conllevan muchas menos operaciones que los dígitos con tantos bits.

### Cambio de base

D'43,2 → Binario.

Separamos la parte entera (43) de la parte decimal.

$$\begin{array}{ll} 43 / 2 = 21 & r = 1 \\ 21 / 2 = 10 & r = 1 \\ 10 / 2 = 5 & r = 0 \\ 5 / 2 = 2 & r = 1 \\ 2 / 2 = 1 & r = 0 \end{array}$$

$$43 = 101011$$

Parte decimal: 0,2

$$0,2 \times 2 = 0,4 \times 2 = 0,8 \times 2 = 1,6 \rightarrow 0,6 \times 2 = 1,2 \rightarrow 0,2 \times 2 = 0,4 \dots$$

$$0,2 = 00110\dots \text{ se repite}$$

$$D'43,2 = 101011,00110\dots$$

No es un número finito, ya que la parte decimal se repite.

Éste método no es eficiente, por lo que hay que pasar directamente de hexadecimal a binario y viceversa.

$$\begin{aligned} 1011100101101011 &= 1011 \ 1001 \ 0110 \ 1011 = \{A \text{ decimal}\} = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \\ 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 &= 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 11 \ 9 \ 6 \ 11 = \\ \{A \text{ hexadecimal}\} &= B \ 9 \ 6 \ B = H'B96B \end{aligned}$$

Para realizar el cambio de base de manera directa, se separan los dígitos del número binario en grupos de cuatro empezando por la derecha y se cambia directamente al valor correspondiente en hexadecimal de cada grupo de cuatro dígitos.

### Operaciones lógicas

|   |   |   |     |  |  |  |  |     |    |     |      |  |  |  |  |     |   |
|---|---|---|-----|--|--|--|--|-----|----|-----|------|--|--|--|--|-----|---|
| a | b | 0 | AND |  |  |  |  | XOR | OR | NOT | XNOR |  |  |  |  | NOT | 1 |
|---|---|---|-----|--|--|--|--|-----|----|-----|------|--|--|--|--|-----|---|

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

Las cuatro operaciones lógicas que se suelen implementar son NOT, AND, OR, XOR.

## Operaciones

- Extensión de signo.
  - Nos permite, partiendo de un número que se representa en N bits, representarlo con M bits  $\rightarrow M > N$
- Desplazamientos.
  - Partiendo de un número  $A = (a_{n-1} a_{n-2} \dots a_1 a_0)$ , tras una operación con desplazamiento, nos saldrá un resultado  $D = (d_{n-1} d_{n-2} \dots d_1 d_0)$
  - El desplazamiento lógico, ya sea izquierda o derecha, se pierde un bit y se rellena con un 0.
  - Si el desplazamiento fuese circular, lo que se cae por el lado al que desplazo se introduce por el lado contrario.
  - Desplazamiento circular conectado con el acarreo.
  - Desplazamiento aritmético: realizan la multiplicación por 2 (a la izquierda) o división por 2 (a la derecha). Dependen de la representación.

|  |  |  |  |  |  |  |  |
|--|--|--|--|--|--|--|--|
|  |  |  |  |  |  |  |  |
|--|--|--|--|--|--|--|--|

|  |  |  |  |  |  |  |  |
|--|--|--|--|--|--|--|--|
|  |  |  |  |  |  |  |  |
|--|--|--|--|--|--|--|--|

## Representación en coma fija

### Binario sin signo

## 1. Representación y valor

Representación (x) =  $(X_{m-1} X_{m-2} \dots X_1 X_0)$

Valor (x) =  $[\text{Sumatorio desde } i = 0 \text{ hasta } n-1] X_i \cdot 2^i$

## 2. Rango y resolución

Rango =  $[0, 2^{n-1} - 1]$

Resolución = 1.

## 3. Cambio de signo

No hay cambio de signo porque estamos en una representación de binario sin signo.

## 4. Extensión de signo

- - - - => X X X X - - - -      ¿ Con que rellenamos las X ?

Da igual lo que tengamos como número, ya que siempre la extensión de signo se realizará añadiendo "0" a la izq del número.

0011 => 0000 0011

## 5. Desplazamiento aritmético

El desplazamiento aritmético se puede realizar de dos formas: a la izquierda del número, entonces será como realizar una multiplicación por 2 al número original o desplazamiento a la derecha que será como realizar una división por 2 al número original.

0110 -> 6 => 1100 -> 12

0110 -> 6 => 0011 -> 3

Pero hay que tener cuidado con la pérdida de bits, ya que perderíamos información del número.

1100 -> 12 => 1000 -> 8

## 6. Suma - Resta

Suma: Se hará sumando como en decimal, teniendo en cuenta que la representación del número solo puede tener dos valores, {0, 1}, así que,  $1+1 \neq 2 = 10$

Resta: sumando el primer número con el complementario del segundo.

$1011 - 0100 \Rightarrow 1011 + 0100 + 1 = 10111 \Rightarrow 0111 + \text{OVF}$

Ha habido OVF, ya que la suma sería D'23 y no lo podemos representar con 4 bits.

Miércoles 6 de Noviembre de 2013

## Complemento a 2

### 1. Representación y valor del complemento a 2

Representación (x) =  $(X_{n-1} X_{n-2} \dots X_1 X_0)$

Si  $X_{n-1} = 0$   $x \geq 0$  Binario sin signo o Binario puro.

$X_{n-1} = 1$   $x < 0$   $|x| = 2^n - \text{Rep}(x) \Rightarrow \text{Rep}(x) = 2^n - |x|$

$\text{Rep}(x) + \text{Rep}(x) = 2^n$

Valor (x) =  $-X_{n-1} \cdot 2^n + [\text{Sumatorio desde } n-1 \text{ desde } i = 0] X_i \cdot 2^i$

Valor (x) =  $-X_{n-1} + -X_{n-1} \cdot 2^{n-1} + [\text{Sumatorio desde } n-2 \text{ desde } i = 0] X_i \cdot 2^i$

Valor (x) =  $-X_{n-1} \cdot 2^{n-1} + [\text{Sumatorio desde } n-2 \text{ desde } i = 0] X_i \cdot 2^i$

Ejemplo:  $n = 6$ ,  $A = 7$ ,  $B = 101110$

$A = 000111$

$-A = 1000000 - 000111 = 111001 = 111000 + 1 = 111001$

(-A se representa invirtiendo los bits de A y sumando 1)

$|B| = 1000000 - 101110 = 010010 = 18$ ,  $B = -18$

Valor máximo =  $011111 = 25 - 1 = 31$

Valor mínimo =  $100000 = -25 = -32$

### 2. Rango y resolución

Tenemos representaciones desde la 0000...00 hasta la 1111...11.

Rango =  $[-2^{n-1}, 2^{n-1} - 1]$

Resolución = 1.

### 3. Cambio de signo

$\text{Rep}(x) + \text{Rep}(-x) = 2^n$

$\text{Rep}(-x) = 2^n - \text{Rep}(x)$

$-A = 2^n - A = (2^n - 1 - A) + 1 = \text{NOT } A + 1$

$010010100 \Rightarrow 101101100$

Invertimos todos los dígitos de izquierda a derecha dejando el último 1 tal cual y dejamos el número tal cual hasta el final del número.

### 4. Extensión de signo

- - - -  $\Rightarrow$  X X X X - - - - ¿Con que rellenamos las X?

Extendemos el valor de la primera cifra

$1000 \Rightarrow 11111000$

$0011 \Rightarrow 00000011$

$1100 \Rightarrow 11111100 \rightarrow$  cambio de signo  $\rightarrow 00000011$

Los números a la izquierda de los números negativos no cambian el valor.



Con  $a=|A|$  siendo  $A < 0$ ,  $2^m - a = (2^n - a) + (2^m - 2^n)$  por lo que hay que rellenar con 1 los  $(m-n)$  bits añadidos.

## 5. Desplazamiento aritmético

Hay que tener cuidado con la pérdida de bits, ya que perderíamos información del número, principalmente en los desplazamientos.

### Desplazamiento a izquierda

0011 = 3  $\rightarrow$  0110 = 6  $\rightarrow$  1100 = 12  $\rightarrow$  OVF

El desplazamiento a la izquierda del número, entonces será como realizar un realizan la multiplicación por 2 al número original.

Cuando el bit de signo que tenía y el que tengo son distintos hay OVF, desbordamiento.

El 12 no se puede representar en 4 bits en completo a 2

### Desplazamiento a la derecha

0101 = +5  $\rightarrow$  0010 = +2 pérdida de bit

1100 = -4  $\rightarrow$  1110 = -2

1101 = -3  $\rightarrow$  1110 = -2 pérdida de bit

Desplazar todos los bits una posición a la derecha y meter un bit de signo (positivos un 0 y negativos un 1).

Desplazamiento a la derecha que será como realizar una división por 2 al número original.

### Suma-resta

|       |            | A         | B         | A+B                   | $C_{n-1}$ | $C_{n-2}$ | OVF              |
|-------|------------|-----------|-----------|-----------------------|-----------|-----------|------------------|
| A+ B+ |            | a         | b         | a + b                 | 0         | -         | 1 si $C_{n-2}=1$ |
| A- B- |            | $2^n - a$ | $2^n - b$ | $2^n - (a + b) + 2^n$ | 1         | -         | 1 si $C_{n-2}=0$ |
| A+ B- | $a \geq b$ | a         | $2^n - b$ | $a - b + 2^n$         | 1         | 1         | 0                |
| A+ B- | $a < b$    | a         | $2^n - b$ | $2^n - (b - a)$       | 0         | 0         | 0                |

### Comparación

A = 0010 = +2

B = 1110 = -2

0010 + 0001 + 1 = 0100      A>B

A = 0110 = +2

B = 1100 = -2

$$0010 + 0001 + 1 = 0100 \quad A > B$$

No se puede utilizar directamente el flag de signo para determinar si es un número mayor que otro, ya que puede haber desbordamiento y el signo está contaminado.

## Complemento a 1

### 1. Representación y valor del complemento a 1

$$\text{Representación (x)} = (X_{n-1} X_{n-2} \dots X_1 X_0)$$

Si  $X_{n-1} = 0$   $x = 0$  Binario sin signo o Binario puro.

$$\begin{aligned} X_{n-1} = 1 \quad x &= 0 & |x| &= 2^n - \text{Rep}(x) \\ & & \text{Rep}(x) &= 2^n - 1 - |x| \end{aligned}$$

$$\text{Valor (x)} = -X_{n-1} \cdot 2^n + [\text{Sumatorio desde } n-1 \text{ desde } i = 0] X_i \cdot 2^i$$

$$\text{Valor (x)} = -X_{n-1} + -X_{n-1} \cdot 2^{n-1} + [\text{Sumatorio desde } n-2 \text{ desde } i = 0] X_i \cdot 2^i$$

$$\text{Valor (x)} = -X_{n-1} \cdot 2^{n-1} + [\text{Sumatorio desde } n-2 \text{ desde } i = 0] X_i \cdot 2^i$$

### 2. Rango y resolución

$$000\dots00 \rightarrow 0$$

$$000\dots01 \rightarrow 1$$

...

$$011\dots11 \rightarrow 2^{n-1}$$

### 3. Cambio de signo

rellenar con unos no cambia el signo

### 4. Extensión de signo

### 5. Desplazamiento aritmético

#### Desplazamiento a izquierda

$$0011 = 3 \rightarrow 0110 = 6 \rightarrow 100 = 12 \rightarrow \text{OVF}$$

Desplazamiento circular de signo.

Operación: Multiplicar por 2

Para detectar desbordamiento: ----

#### Desplazamiento a la derecha

Desplazamiento circular de signo.

Operación: dividir por dos

Para detectar desbordamiento: ----

11 / NOV / 2013

### Ejercicio

1. Rango y representación
2.  $A = -5'59$   
 $B = +4'25$   
 $C = -3'75$
3.  $A - B - C$
4.  $A - 2 \cdot C$

| Número    | Complemento a 2 | Complemento a 1   |
|-----------|-----------------|-------------------|
| 0000,0000 | 0               | 0                 |
| 0000,0001 | $2^{-4}$        | $2^{-4}$          |
| ...       | ...             | ...               |
| 0111,1111 | $2^3 - 2^{-4}$  | $2^3 - 2^{-4}$    |
| 1000,0000 | $-2^3$          | $-(2^3 - 2^{-4})$ |
| ...       | ...             | ...               |
| 1111,1111 | $-2^{-4}$       | -0                |

$A = -0101,1001$   
 $C1 = 1010,0110$   
 $C2 = 1010,0111$

$B = 0100,0100$   
 $C1 = 0100,0100$   
 $C2 = 0100,0100$

$C = -0011,1100$   
 $C1 = 1100,0011$   
 $C2 = 1100,0100$

### Complemento a 2

11010,0111  
 11011,1011  
 1

-----

10110,0011  
 00011,1011  
 1

-----

41001,1111

$A - B - C = 1001,1111 = -0110,0001 = -6,0625$   
 $-5'59 - 4'25 = -9,84 + 3'75 = -6'09$

### Complemento a 1

11010,0110

$A - B - C = 1001,1110 = -0100,0001$

```

11011,1011

110110,0001
 1

10110,0010
00011,1100

41001,1110

```

A - 2\*C

C a 2C a 1

C = 1100,0100  
 2C= 1000,1000

C = 1100,0011  
 2C= 1000,0111

```

1010,0111
0111,0111
 1

```

0001,1111

A - 2C = 1'9375

```

1010,0110
0111,1000

10001,1110
 1

```

0001,1111

A - 2C = 0001,1111

26 / NOV / 2013

## TEMA 5 - PERIFÉRICOS

**FALTA LO PRIMERO**