Tema 2 : Arquitectura Von Neumann

HISTORIAL DE REVISIONES

NÚMERO	FECHA	MODIFICACIONES	NOMBRE
0.6.0	localdate		C.

Índice

1. Arquitectura Von Neumann

1.1. Temario

1. Arquitectura Von Neumann:
    a. CPU
    b. Memoria
    c. Entrada / Salida

1.2. Contexto Histórico

1.2.1. Antecedentes

- 1833: Charles Babbage → Diseña la 1ª Computadora mecánica
- 1890: Máquina tabuladora de Herman **Hollerith**. Censo en USA. IBM (1925)
- 1936: Alan Turing → Algoritmia y concepto de máquina de Turing. Máquina código Enigma.
- **Seguna Guerra Mundial 1939-1945**
- 1944: USA, IBM Computadora electromecánica Harvard Mark I
- 1944: Colossus (Colossus Mark I y Colossus Mark 2). Decodificar comunicaciones.

1.2.2. ENIAC

- 1947: En la Universidad de Pensilvania (laboratorio de investigación de balística para la artillería) se construye la **ENIAC** (Electronic Numerical Integrator And Calculator)
    - Ecuaciones diferenciales sobre balística (angle = f (location, tail wind, cross wind, air density, temperature, weight of shell, propellant charge, ...))
    - Computadora electrónica (no mecánica) de **propósito general**.
    - Memoria: Sólo 20 acumuladores → flip-flops hechos con triodos
        - 18,000 tubos electrónicos ó válvulas de vacío
    - Programación manual de los interruptores
    - 100,000 instrucciones por segundo
    - 300 multiplicaciones por segundo
    - 200 kW
    - 13 toneladas y 180 m^2

1.2.3. EDVAC

- 1951: En la Universidad de Pensilvania (J. Presper Eckert y John William Mauchly) comienza a operar la **EDVAC** (Electronic Discrete Variable Automatic Computer), concebida por **John von Neumann**, que a diferencia de la ENIAC no era decimal, sino binaria, y tuvo el primer **programa** (no solo los datos) diseñado para ser **almacenado**: STORED PROGRAM COMPUTER → program can be manipulated as data.
    - 500000$
    - La EDVAC poseía físicamente casi 6000 válvulas termoiónicas y 12 000 diodos de cristal. Consumía 56 kilowatts de potencia. Cubría 45,5 m^2 de superficie y pesaba 7850 kg.

- Arquitectura:
  - un lector-grabador de cinta magnética
  - una unidad de control con osciloscopio, una unidad para recibir instrucciones del control
  - la memoria : 2000 word storage "mercury delay lines" → poca fiabilidad
  - una unidad de aritmética de coma flotante en 1958.

1.2.4. IAS

- 1946-1952 : **IAS** (Institute Advanced Studies) mainframe :
  - Evolución de EDVAC: unidad de memoria principal y secundaria tambor magnético.
  - Memoria Selectron: almacenamiento capacitivo → carga electrostática

1.2.5. Posterior

- 1952: **UNIVAC I** (UNIVersal Automatic Computer I) was the first commercial mainframe computer. Evolución de la máquina tabuladora de Hollerith aplicado al procesado del censo en USA.
- 1952: IBM 701, conocido como la "calculadora de Defensa" mientras era desarrollado, fue la primera computadora científica comercial de IBM → primer lenguaje **ENSAMBLADOR**.
- 1964: mainframe (computadora central) **IBM 360** → primer computador con ISA (microprogramación) → compatibilidad
  - tecnología híbrida entre componentes integrados discretos de silicio y otros componentes → no "circuitos" integrados.
  - Basic Operating System/360 (BOS/360), Disk Operating System/360 (DOS/360)

1.2.6. Tecnología de Semiconductor

- 1947: en los Laboratorios Bell, John Bardeen, Walter H. Brattain y William Shockley inventan el **transistor**.
- 1958: Kilby , primer circuito integrado en germanio.
- 1957: Robert Norton Noyce, cofundador de Fairchild Semiconductor, primer circuito integrado planar
- 1968: Robert Norton Noyce y Gordon Moore fundan Intel.
- 1971: Intel 4004 → cpu integrada en silicio → 8 bits

2. Institute Advanced Machine (IAS) : Arquitectura

2.1. Referencia

- The Von Neumann Machine

2.2. Ejemplo del Programa sum1toN

2.2.1. Código

./images/von_neumann/ias_code_machine.png

Figura 1: IAS Codigo Maquina

2.2.2. Programación Imperativa

- Paradigma:
  - Paradigma imperativo ó estructural : el algoritmo se implementa desarrollando un programa que contiene las ORDENES que ha de ejecutar la máquina
    - A diferencia de la programación declarativa: el algoritmo implementa QUÉ queremos que haga la computadora, no el COMO, no directamente las órdenes que ha de ejecutar.
    - Por ejemplo $\sum_{i=1}^{5} i$ en python:

```
>>> sum(range(5,0,-1))
```

  - La computadora IAS se programaba directamente en *lenguaje máquina*, no tenía un lenguaje simbólico como el lenguaje ensamblador.
  - Lenguaje Máquina: Código Binario
  - Edición del código binario mediante tarjetas perforadas o cintas magnéticas a través de una consola.
  - Tipo de información : INSTRUCCIONES Y DATOS
    - Ejemplo de instrucciones: inicializar un dato, mover un dato, sumar datos, saltar a una instrucción determinada, etc
- Concepto de progama **almacenado** : Instrucciones binarias y Datos binarios almacenados en la **Unidad de Memoria**
  - Es necesario CARGAR el módulo binario en la MEMORIA de la computadora.
- Programación secuencial: Las instrucciones se ejecutan secuencialmente según están almacenadas en la memoria. . . mientras no se ejecute una instrucción de salto.

2.2.3. Arquitectura

- Para poder analizar el programa es necesario no solo conocer el lenguaje máquina sino conocer su ARQUITECTURA. La arquitectura de una compuradora es el WHAT de la máquina, es decir, QUE instrucciones es capaz de ejecutar la máquina, para lo cual es necesario conocer la ARQUITECTURA DEL REPERTORIO DE INSTRUCCIONES (Instruction Set Architecture **ISA**):
  - el repertorio de instrucciones: operaciones y modo de acceso a los datos
  - jerarquía de memoria: memoria principal y registros
  - formato de instrucciones y datos

3. IAS: Estructura

3.1. Módulos

---

nota
La Estructura es el HOW de la máquina. De qué hardware disponemos para poder ejecutar las instrucciones máquina definidas por la arquitectura.

- Hardware con Estructura **Modular**:
  - CPU-Memoria-I/O-Bus
    - Jerarquía de Memoria: 2 niveles : Memoria Principal (externa a la CPU) y Registros (internos a la CPU)

./images/von_neumann/ias_architecture.png

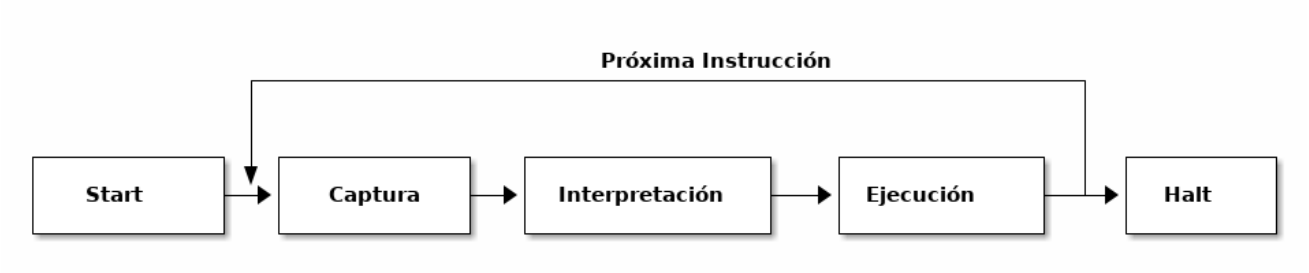
Figura 2: IAS_Architecture

- Arquitectura Interna de la CPU : Microarquitectura

Figura 3: IAS Structure

3.2. CPU

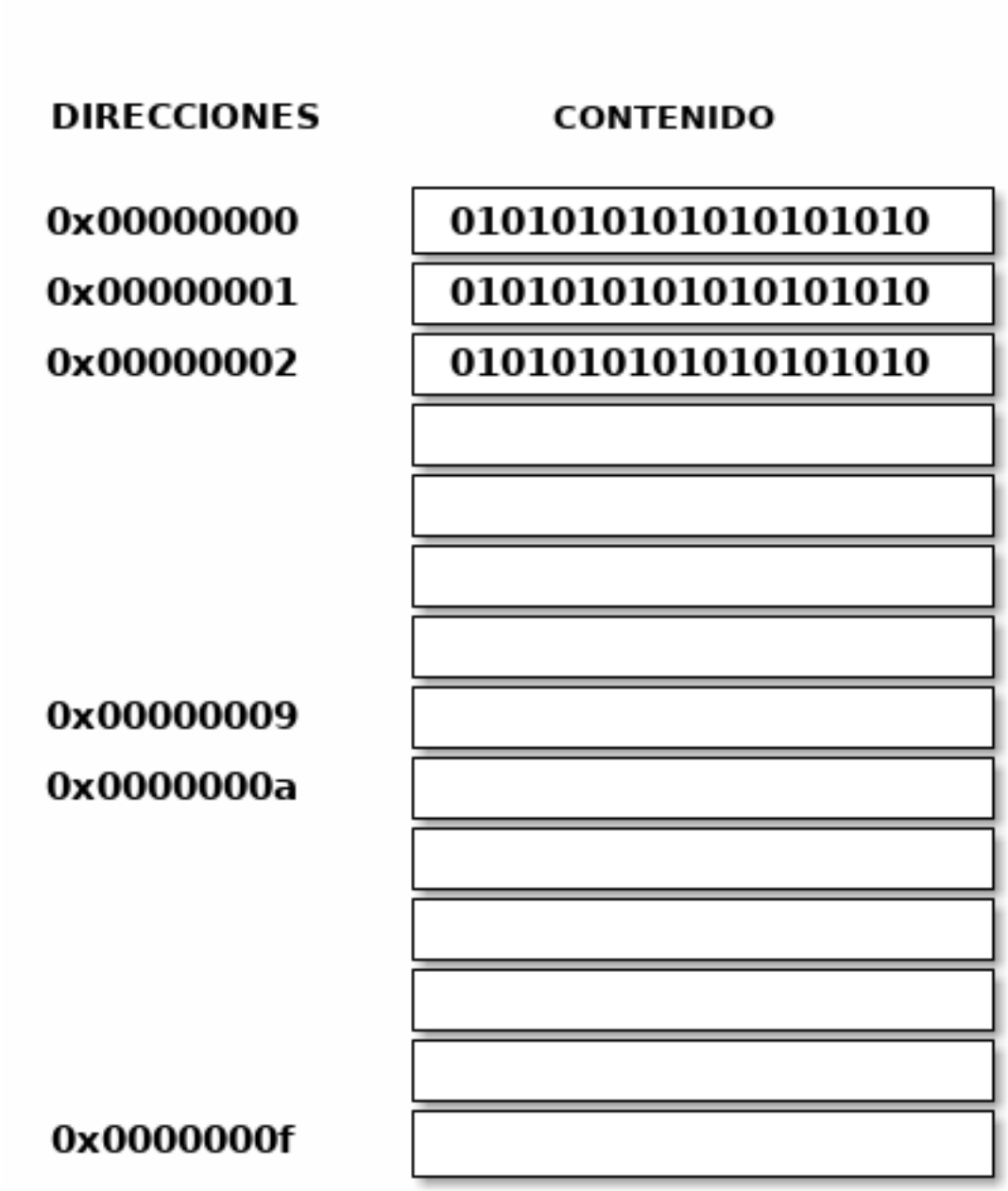
- CPU:
  - 3 FASES: Captura, Interpreta y Ejecuta las instrucciones secuencialmente: **Ciclo de Instrucción**



- Tres submódulos principales de la CPU:
  - Unidad de cálculo: Unidad Aritmético-Lógica (ALU)
  - Unidad de control: Circuito secuencial que implementa el Ciclo de instrucción dando las órdenes eléctricas a los distintos bloques (ALU, memoria principal, registros, buses, etc) en cada fase hasta completar el ciclo.
  - Registros

3.3. Memorias

3.3.1. Memoria Principal



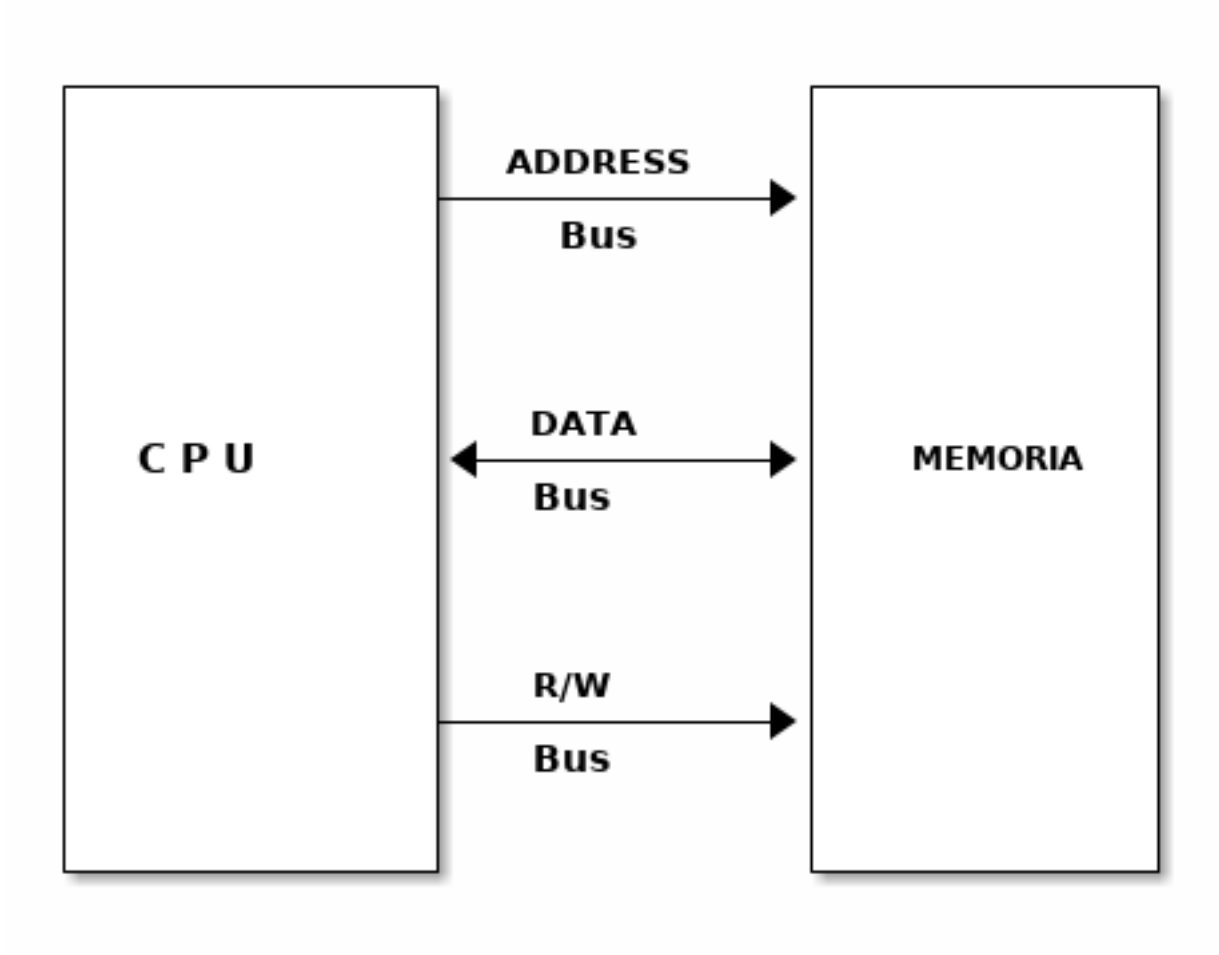
- Memoria Principal
  - Debe almacenar el programa a ejecutar en código binario.
  - La CPU es el único módulo que tiene acceso a la memoria principal.
  - Las instrucciones y datos del programa se almacenan secuencialmente.
  - Almacena el programa en dos *secciones*: Sección de Datos y Sección de Instrucciones
  - Organizada en Palabras accesibles aleatoriamente. Random Access Memory.
  - Dinamismo: Lectura/Escritura de datos e instrucciones

3.3.2. Registros

- Registros:
  - Memoria interna a la CPU: GPR más el resto de registros no transparente al programador
  - General Register Purpose (GPR) (accesibles por el programador): *AC* y *AR/MQ*.
  - Selectron Register (no accesible por el programador): Memory Buffer Register *MBR*. Data Buffer Register *DBR*. 40 bits.
  - Registros NO accesibles por el programador: todos los registros de la Unidad de Control: PC,IR,MAR,IBR
    - Control Counter: Program Counter *PC* o Instruction Pointer IP. 12 bits. Apunta a la siguiente instrucción a capturar
    - Control Register: Instruction Register *IR*. 8 bits.
    - Instruction Buffer Register: *IBR*. Siguiente Instrucción a Ejecutar. 20 bits
    - Memory Address Register: *MAR* , current Memory Address. 12 Bits. Apunta al operando o instrucción a capturar.

3.4. Bus

- Bus del Sistema:
  - Interconexión CPU-Memoria Principal: transferencia de datos e instrucciones.
  - Bus de Datos, Bus de Direcciones y Bus de Control (Lectura/Escritura)

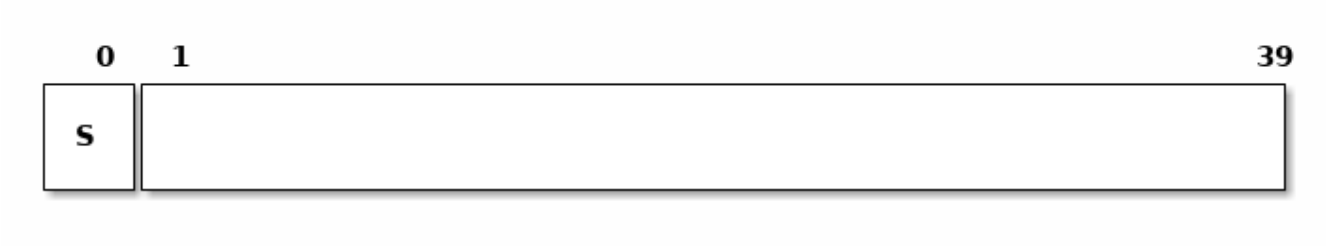


3.5. I/O

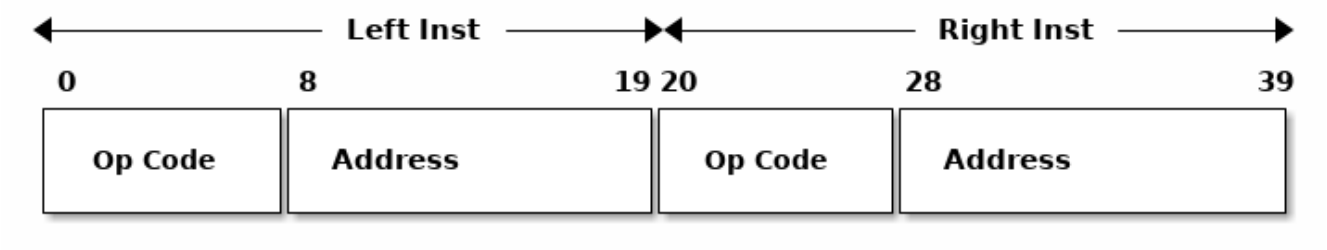
- I/O
  - El programa se escribe en tarjetas perforadas (Punch Cards). Tarjetas para Datos y tarjetas para instrucciones. Es necesario cargar los datos en la memoria antes de la ejecución del programa.
  - tarjetas perforadas, consola, tambores magnéticos, cintas magnéticas, cargador de memoria mediante un lector de tarjetas , display mediante tubos de vacío, etc.. $\rightarrow$ tecnología obsoleta.
  - No tendremos en cuenta el módulo I/O y nos centraremos en los módulos CPU-Memoria Principal.

4. IAS: Formato de los datos e Instrucciones

- Arquitectura de la Memoria
  - Word
    - 40 bits : 1 dato ó 2 instrucciones
  - Datos
    - Entero: Complemento a 2
  - Instrucciones
    - 8-bit opcode followed by a 12-bit operand (data address)
  - Memory
    - Random Access Memory: direccionable cada posición de memoria.
    - Almacena el programa: datos e instrucciones
      - Shared Memory: memoria compartida entre datos e instrucciones. También comparten el bus de acceso a memoria.
    - Capacidad para 2^12=4K palabras
      - 4K x 40bits = 4K x 5Bytes = 20KBytes
      - En cambio la memoria física disponible en esa época era de : 1024 palabras de 40 bits = 5 KBytes (Libro "The Computer from Pascal to von Neumann", Herdman Godstine, pg314, ISBN 0-691-02367-0)
  - 7 Registros
    - Acumulator: AC: Visible al Programador. 40 Bits
    - Arithmetic Register AR, también llamado registro Multiplier/Quotient MQ: Visible al programador. 40 Bits
- *Data Format*



  - Observar que el bit con la numeración cero es el de la izda.
- *Instruction Format*



  - Definimos **un sólo operando** o ninguno en cada instrucción
    - *Accumulator Based Architecture*
    - Una operación que requiera dos operandos implicitamente hace referencia a un operando almacenado en el *acumulador*

- Observar que el bit con la numeración cero es el de la izda.
- La instrucción de la izda (0-19) se carga en los registros, internos de la CPU, el código de operación IR y el campo de operación en MAR .
- La instrucción de la derecha (20-39) se carga en el registro, interno de la CPU, IBR .
- Modo de direccionamiento del Operando
  - Referencia del Operando. Dirección de Memoria.

- Contenido de la Memoria
  - Las direcciones de memoria las visualizamos dobles ya que hacen referencia a la primera a los 20 bits LSB y la segunda a los 20 bits MSB de una palabra de memoria de 40 bits.
  - Observar que en la columna data están las dos secciones: sección de instrucciones y sección de datos
  - En la arquitectura von Neumann datos e instrucciones comparten el mismo espacio de direcciones de memoria.

./images/von_neumann/ias_code_machine.png

Figura 4: IAS Codigo Maquina

5. Arquitectura del Repertorio de Instrucciones de la máquina IAS

5.1. Repertorio ISA

- Instruction Set Architecture (ISA): Definición y características del conjunto de instrucciones. Arquitectura del Repertorio de Instrucciones.
- En la versión original no había código ensamblador, se programaba directamente en lenguaje máquina.
  - En la tabla adjunta, en la segunda columna, los **MNEMONICOS** (LOAD,ADD,SUB,etc) de las operaciones de las instrucciones se corresponden con los diseñados por el libro de texto de William Stalling. En la primera y última columnas las operaciones se simbolizan mediante un lenguaje de transferencia entre registros.
  - Selectron es el nombre de la tecnología utilizada para la Memoria Principal.
    - La notación S(x) equivale en notación RTL a M[x]
  - R es el registro AR que W.Stalling denomina registro MQ.

Cuadro 1: Instruction Set

Instruction name	**Instruction name**	**Opcode**	**Description**	**RTL**
S(x)→Ac+	LOAD M(X)	1	copy the number in Selectron location x into AC	AC ← M[x]
S(x)→Ac−	LOAD −M(X)	2	same as #1 but copy the negative of the number	AC ← ~M[x]+1
S(x)→AcM	LOAD |M(X)|	3	same as #1 but copy the absolute value	AC ← |M[x]|
S(x)→Ac−M	LOAD −|M(X)|	4	same as #1 but subtract the absolute value	AC ← AC−|M[x]|
S(x)→Ah+	ADD M(X)	5	add the number in Selectron location x into AC	
S(x)→Ah−	SUB M(X)	6	subtract the number in Selectron location x from AC	
S(X)→AhM	ADD |M(X)|	7	same as #5, but add the absolute value	
S(X)→Ah−M	SUB |M(X)|	8	same as #7, but subtract the absolute value	
S(x)→R	LOAD MQ,M(X)	9	copy the number in Selectron location x into AR	
R→A	LOAD MQ	A	copy the number in AR to AC	
S(x)∗R→A	MUL M(X)	B	Multiply the number in Selectron location x by the number in AR. Place the left half of the result in AC and the right half in AR.	
A/S(x)→R	DIV M(X)	C	Divide the number in AC by the number in Selectron location x. Place the quotient in AR and the remainder in AC.	
Cu→S(x)	JUMP M(X,0:19)	D	Continue execution at the left-hand instruction of the pair at Selectron location x	
Cu`→S(x)	JUMP M(X,20:39)	E	Continue execution at the right-hand instruction of the pair at Selectron location x	
Cc→S(x)	JUMP+ M(X,0:19)	F	If the number in AC is >= 0, continue as in #D. Otherwise, continue normally.	

Cuadro 1: (continued)

Instruction name	**Instruction name**	**Opcode**	**Description**	**RTL**
Cc`→S(x)	JUMP+ M(X,20:39)	10	If the number in AC is >= 0, continue as in #E. Otherwise, continue normally.	
At→S(x)	STOR M(X)	11	Copy the number in AC to Selectron location x	
Ap→S(x)		12	Replace the right-hand 12 bits of the left-hand instruction at Selectron location x by the right-hand 12 bits of the AC	
Ap`→S(x)		13	Same as #12 but modifies the right-hand instruction	
L	LSH	14	Shift the number in AC to the left 1 bit (new bit on the right is 0)	
R	RSH	15	Shift the number in AC to the right 1 bit (leftmost bit is copied)	
halt		0	Halt the program (see paragraph 6.8.5 of the IAS report)	

- Instruction Set (William Stalling)

./images/von_neumann/ias_instruction.png

Figura 5: IAS_Instruction_Set

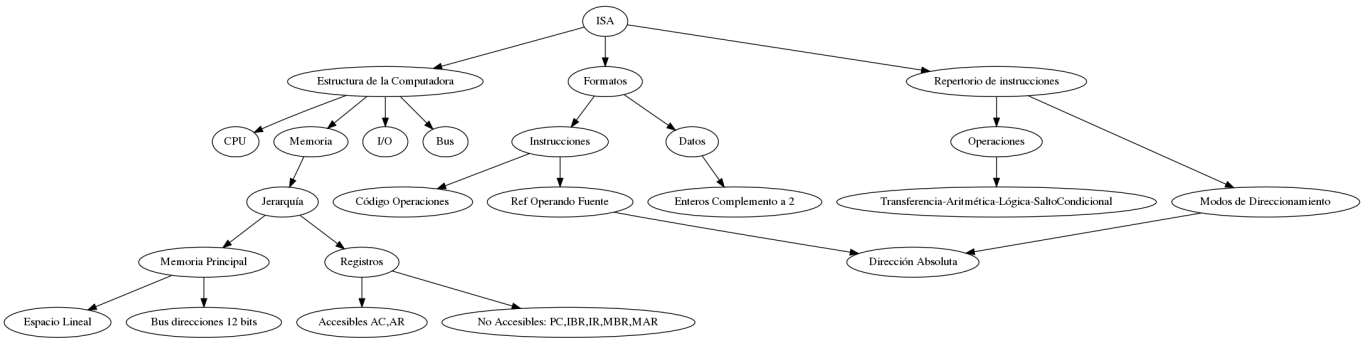
5.2. Interfaz ISA

- La arquitectura del conjunto de instrucciones (ISA) define la **INTERFAZ** entre el Hardware y el Software de la máquina
  - Podemos tener dos CPU totalmente diferentes, p.ej AMD e Intel, pero si tienen la misma ISA serán máquinas compatibles desde el punto de vista del sistema operativo.
  - Concepto de familia: un mismo repertorio de instrucciones puede ser ejecutado por distintas computadoras
- ISA de distintas máquinas

6. Recordatorio

- Ideas Fundamentales
- PROGRAMACION DE BAJO DE NIVEL
  - Programación con *acceso directo* a los recursos HW de la computadora.
- ARQUITECTURA Von Neumann:
  - Codificación de las Instrucciones
  - Programa Almacenado (datos e instrucciones comparten la misma unidad de memoria)
- ARQUITECTURA ISA:
  - Estructura (CPU/MEMORIA/I-0)
  - Formato Datos e Instrucciones
  - Repertorio de instrucciones (Operaciones y Modos de Direccionamiento)
  - Etc...
- **ISA-IAS**: Arquitectura ISA de la máquina IAS

7. ISA


ISA
Estructura de la Computadora
Formatos
Repertorio de instrucciones
CPU
Memoria
I/O
Bus
Instrucciones
Datos
Operaciones
Jerarquía
Código Operaciones
Ref Operando Fuente
Enteros Complemento a 2
Transferencia-Aritmética-Lógica-SaltoCondicional
Modos de Direccionamiento
Memoria Principal
Registros
Dirección Absoluta
Espacio Lineal
Bus direcciones 12 bits
Accesibles AC,AR
No Accesibles: PC,IBR,IR,MBR,MAR

8. Programación en el Lenguaje Ensamblador IAS

8.1. Máquina Virtual Java JVM

- Instalar el Kit de Desarrollo Java (Java Development Kit-JDK) en el sistema ubuntu
  - openjdk-11-jdk en la distribución linux/GNU ubuntu 18.0 bionic.
  - Comprobar que se tiene acceso al paquete: `apt-cache search openjdk-11-jdk`
  - Instalar el paquete: `sudo apt-get install openjdk-11-jdk`
  - Comprobar que está instalado el paquete: `dpkg -l openjdk-11-jdk`
  - Comprobar la versión de java instalada: `java --version`
- datos de la instalación en Ubuntu 17

```
Date: September 15, 2017.
Emulator version: IASSim2.0.4
Emulator command: java -cp IASSim2.0.4.jar:jhall.jar:IASSimHelp2.0.jar iassim.Main -m IAS. ↩
    cpu
Operating System: GNU/linux
        Distributor ID: Ubuntu
        Description:    Ubuntu 17.04
        Release:        17.04
        Codename:       zesty
Java version: openjdk version "1.8.0_131"
              OpenJDK Runtime Environment (build 1.8.0_131-8u131-b11-2ubuntu1.17.04.3-b11)
              OpenJDK 64-Bit Server VM (build 25.131-b11, mixed mode)
```

IASSim se ejecuta en la máquina virtual de Java JVM, por lo tanto es un requisito tener instalada la máquina virtual JVM. Virtualizar una máquina consiste en instalar una capa SW por encima de cualquier Sistema Operativo (Linux, MacOS, Windows) de tal forma que cualquier aplicación (Por ejemplo IASSim) que se instale sobre la capa de virtualización no depende del Sistema Operativo y así se consigue independizar la aplicación (Por ejemplo IASSim) de los diferentes Sistemas Operativos.

8.2. Simulador IAS

- IASSim : Herramienta de simulación de la computadora IAS de Von Neumann útil para la simulación de la ejecución paso a paso de las instrucciones de un programa en código máquina. Permite visulalizar el contenido de la memoria principal Selectron y de los registros de la CPU al finalizar cada ciclo de instrucción.
- IASSim Web : Al hacer click nos conectamos al repositorio del simulador IASSim.
  - Descargar el Simulador IASSim2.0.4 : archivo zip
  - Instalar el simulador IASSim2.0.4 (descomprimir el archivo zip).
- Abrir el Simulador:

1. ../IASSim2.0.4$ `java -cp IASSim2.0.4.jar:jhall.jar:IASSimHelp2.0.jar iassim.Main -m IAS.cp`

8.2.1. Registros

- The IAS machine has 7 registers: Accumulator, Arithmetic Register / Multiplier-Quotient (AR/MQ), Control Counter, Control Register, Function Table Register, Memory Address Register, Selectron Register
  - The Accumulator (*AC*) and Arithmetic registers (*AR/MQ*) are the only two programmer-visible registers
  - The Control Counter is what we now call the Program Counter *PC*
  - The Control Register holds the currently executing instruction *IBR*. Unicamente la instrucción de la derecha que se va a ejecutar.
  - The Function Table Register holds the current opcode *IR*
  - The Memory Address Register the current memory address *MAR*
  - Selectron Register the current data value being read from or written to memory $\rightarrow$ *MBR*

8.2.2. Notas

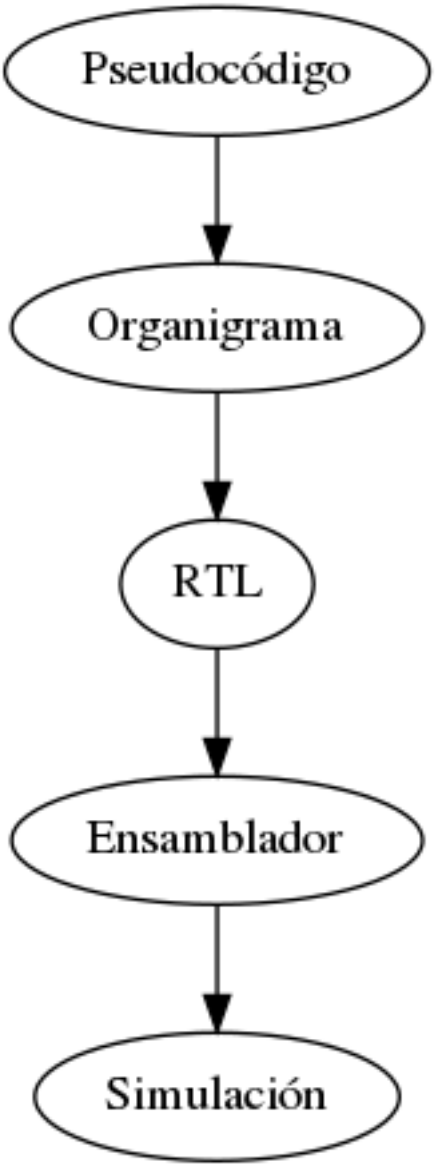
- Es necesario que el número de instrucciones sea par. Si es impar se añade la directiva *.empty*.
- Una etiqueta debe de apuntar a la instrucción izda. Si está en la dcha se puede anteponer una instrucción de salto incondicional a dicha etiqueta.
- La sección de datos si está a continuación de la sección de código hay que terminar la sección de código con una instrucción en la dcha y si no la rellenamos con la directiva *.empty*.

8.2.3. Error

- Error al visualizar el valor del registro MAR
  - Al ejecutar la primera instrucción de sum1toN.ias el contenido de MAR es 28, mayor que el rango de direcciones de la memoria principal donde esta cargado el programa.
  - El error se da tanto en Windows 7 como en Ubuntu 17.04

8.3. Estrategia del Desarrollo de un Programa en Lenguaje Ensamblador

- Una vez entendido el problema que ha de resolverse mediante programación, no se programa directamente el módulo fuente solución del problema sino que se va resolviendo describiendo el problema y el algorimto solución en distintos lenguajes y en las siguiente fases:
  - Descripción del algoritmo en lenguaje "pseudocódigo".
  - Descripción del algoritmo mediante un organigrama o diagrama de flujo.
  - Descripción del algoritmo en lenguaje de transferencia entre registros RTL.
  - Descripción del algoritmo en lenguaje ensamblador iassim



importante
El paso de una descripción en un lenguaje de alto nivel a bajo nivel se realiza en lenguaje RTL teniendo en cuenta la arquitectura de la computadora donde se ejecutará el lenguaje máquina. Cada instrucción de alto nivel habrá que traducirla en un bloque de instrucciones de bajo nivel

8.4. Tutorial1: sum1toN.ias

8.4.1. Enunciado

- Calcular la suma $\sum_{i=1}^{N} i = N(N+1)/2$

8.4.2. Pseudocódigo

- Descripción del algoritmo mediante expresiones modo texto en lenguaje NATURAL

```
* Reservar MEMORIA:
** variable suma : almacena los resultados parciales y final
** variable N    : almacena el dato de entrada
* Estructura del CODIGO imperativo:
** Bucle
*** Genera los sumandos i en sentido descendente -> i = N,N-1,...,1
*** Realiza la suma parcial suma=suma+i
*** contador de iteraciones j desde j=1 hasta j=N
```

8.4.3. Organigrama

Diagrama de Flujo

8.4.4. RTL

- Descripción RTL para la máquina IAS orientada a acumulador

```
;CABECERA
;Descripcion en lenguaje RTL del algoritmo sum1toN

;SECCION DATOS  :
; Declaracion de etiquetas, reserva de memoria externa, inicializacion
; Variables ordinarias
n:      M[n]    <- 5            ; variable sumando e inicializacion
suma:   M[suma] <- 0            ; variable suma parcial y final

;SECCION INSTRUCCIONES
;Arquitectura orientada a Acumulador (AC)
;Registros accesibles : AC
        ; inicio bucle : suma y generación de sumandos
bucle:  AC      <- M[n]         ; cargar sumando
        AC>=0 : PC <- sumar     ; si el sumando < 0 fin del bucle
        ; fin del bucle
        stop
        ; realizar la suma
sumar:  AC      <- AC + M[suma]
        M[suma] <- AC
        ; actualizar sumando
        AC      <- M[n]
        AC      <- AC - 1
        M[n]    <- AC
        ; siguiente iteracion
        PC      <- bucle
```

8.4.5. Lenguaje ensamblador iassim

El desarrollo del módulo fuente en lenguaje ensamblador iassim NO se realiza de principio a fin sino que se va realizando **POR PASOS**, empezando por un código lo más sencillo posible que será testeado y depurado antes de ir desarrollando hasta llegar al código completo

- **1ª Versión**: módulo fuente *sum1toN_v1.ias*:
  - La 1ª versión implementa un bucle cuyo cuerpo únicamente almacena un dato en la variable suma. El dato varía en cada iteración.
  - Sintaxis → etiqueta: operacion operando ;comentario → 4 Columnas
  - Los símbolos para indicar la operación (Ej. S(x)→Ac+) no son mnemónicos
  - No utilizar tildes ni en los comentarios ni en las etiquetas, ya que únicamente se admite código ASCII no extendido.
  - Si el número de instrucciones es impar se ha de rellenar la palabra de 40 bits de la última instrucción con los 20 bits de menor peso a cero.
  - La sección de instrucciones debe de ir previamente a la sección de datos

```
; CABECERA
; 1ª version : sum1toN_v1.ias
; Calcula la suma de una secuencia de numeros enteros: suma = 1+2+..+n
; dato de entrada : n
; dato de salida  : suma
; Algoritmo : bucle de n iteraciones
;             Los sumandos se van generando en sentido descendente de n a 0
;             Se sale del bucle si el sumando es negativo -> -1
; Estructuras de datos : variables n y suma . Constante uno.
; Lenguaje ensamblador: IASSim
; Arquitectura de la máquina IAS de Von Neumann

;;;;;;;;;;;;;;;; SECCION DE INSTRUCCIONES
;Arquitectura orientada a Acumulador (AC)
;Registros accesibles : AC
; algoritmo: bucle que genera la secuencia n, n-1, n-2,...0,-1 si n>=0
bucle:  S(x)->Ac+  n            ; AC <- M[n]
        S(x)->Ah-  uno          ; AC <- AC-M[uno]
        At->S(x)   suma         ; M[suma] <- AC
        Cc->S(x)   bucle        ; Si AC >= 0, salto a bucle
        ; fin del bucle
        halt                    ; stop
        .empty

;;;;;;;;;;;;;;;;;;;SECCION DE DATOS
; Declaracion de etiquetas, reserva de memoria externa, inicializacion.
; Variables ordinarias
n:      .data   5 ; variable sumando
uno:    .data   1 ; cte
suma:   .data   0 ; sumas parciales y resultado final
```

  - Se ha desarrollado la sección de datos para la reserva de memoria.
  - Se ha realizado un BUCLE SENCILLO ya que el bucle es la construcción necesaria en el algoritmo final.
  - Se ha realizado la operación RESTA ya que es una operación necesaria en el algoritmo final.
  - Se ha COMENTADO el código

- **2ª Versión**: módulo fuente *sum1toN_v2.ias*:
  - La 2ª versión implementa un bucle cuyo cuerpo realiza una suma parcial donde uno de los sumandos varía en cada iteración.

```
;;;;;;;;;;;;;; CABECERA
; ; 2ª version : sum1toN_v2.ias
;;;;;;;;;;;;;;;; SECCION DE INSTRUCCIONES
;Arquitectura orientada a Acumulador (AC)
;Registros accesibles : AC
; algoritmo: Bucle que realiza la operación suma = n + suma si n>=0
; inicio bucle
bucle:  S(x)->Ac+  n    ; AC <- M[n] .Cargar sumando
        Cc->S(x)   sumar  ; si AC >= 0, PC <- sumar .Si el sumando < 0 fin del bucle
        halt            ; stop
        .empty          ; un 20-bit 0, para que el nº de instrucciones sea par.
        ; realizar la suma
sumar:  S(x)->Ah+  suma ; AC <- AC + M[suma]
        At->S(x)   suma ; M[suma] <- AC

;;;;;;;;;;;;;;;;;;;SECCION DE DATOS
; Declaracion de etiquetas, reserva de memoria externa, inicializacion.
; Variables ordinarias
n:      .data   5 ; variable sumando
uno:    .data   1 ; cte
suma:   .data   0 ; sumas parciales y resultado final
```

  - Se ha desarrollado la sección de datos.
  - Se ha realizado un BUCLE con la operación SUMA en el cuerpo del bucle y con PARADA al salirse del bucle.
  - Se ha COMENTADO el código

- **Versión Demo** *tutorial.ias*: la versión demo que se incluye en el archivo de descarga del simulador.

```
loop:   S(x)->Ac+  n    ;load n into AC
        Cc->S(x)   pos  ;if AC >= 0, jump to pos
        halt            ;otherwise done
        .empty          ;a 20-bit 0
pos:    S(x)->Ah+  sum  ;add n to the sum
        At->S(x)   sum  ;put total back at sum
        S(x)->Ac+  n    ;load n into AC
        S(x)->Ah-  one  ;decrement n
        At->S(x)   n    ;store decremented n
        Cu->S(x)   loop ;go back and do it again
n:     .data 5  ;will loop 6 times total
one:   .data 1  ;constant for decrementing n
sum:   .data 0  ;where the running/final total is kept
```

- Ejercicio con la **Versión Demo** *tutorial.ias*:
  - cambiar el nombre del módulo fuente: *sum1toN.ias*
  - reeditar el programa con etiquetas y comentarios en castellano.
  - comentar el código con la información de los módulos descritos en las fases previas del desarrollo del programa

```
;;;;;;;;;;;;;; CABECERA
; Modulo fuente sum1toN.ias
; Calcula la suma de una secuencia de numeros enteros: suma = 1+2+..+n
; dato de entrada : N y dato de salida : suma
; Algoritmo : bucle de N iteraciones
;             Los sumandos se van generando en sentido descendente de n a -1
;             Si el sumando es negativo -> -1 , no se realiza la suma y finaliza el  ↩
    bucle
; Estructuras de datos : variables n y suma . Constante uno.
; Lenguaje ensamblador: IASSim
; ISA: Arquitectura de la maquina IAS de Von Neumann

;;;;;;;;;;;;;;;; SECCION DE INSTRUCCIONES
;Arquitectura orientada a Acumulador (AC)
;Registros accesibles : AC
; algoritmo: Bucle que genera los sumandos n, n-1, .... -1
;            y realiza la operación suma = n + suma si n>=0

; inicio bucle : suma y generacion de sumandos
bucle:  S(x)->Ac+  n            ;cargar sumando
        Cc->S(x)   sumar        ;si el sumando < 0 fin del bucle
        ; fin del bucle
        halt                    ; stop
        .empty                  ;a 20-bit 0 para que el nº de instrucciones sea par.
        ; realizar la suma
sumar:  S(x)->Ah+  sum  ;
        At->S(x)   sum  ;
        ; actualizar sumando
        S(x)->Ac+  n    ;
        S(x)->Ah-  uno  ;
        At->S(x)   n    ;
        ; siguiente iteracion
        Cu->S(x)   bucle ;

;;;;;;;;;;;;;;;;;;SECCION DE DATOS
; Declaracion de etiquetas, reserva de memoria externa, inicializacion.
; Variables ordinarias
n:     .data 5  ; sumando e inicializacion
sum:   .data 0  ; suma parcial y final
```

```
; constantes
uno:    .data 1  ;
```

8.4.6. Simulación/Depuración

- Los objetivos de la simulación son dos:
  a. Interpretar la ejecución de cada instrucción observando como varía la memoria y los registros
  b. Depurar posibles errores en el desarrollo del programa.
- https://www.linuxvoice.com/john-von-neumann/
- Es necesario conocer la codificación hexadecimal de los números enteros y su conversión a código binario.
- Al programa *tutorial.ias* le llamaremos *sum1toN.ias*
  1. Borrar el contenido de la memoria tanto interna como externa. `Execute → Clear all`
  2. Desactivar el modo depuración : `Execute → Debug Mode NO seleccionado`
  3. Cargar el programa *sum1toN.ias* en lenguaje ensamblador : `File → Open → sum1toN.ias`
     - Lenguaje ensamblador: creado por los autores de la aplicación *IASSim*.
  4. Ventana RAM Selectrons: direcciones y contenido en código hexadecimal,decimal,binario... Anchura memoria : 20 ó 40 bits.
  5. Seleccionar la ventana con el código fuente en lenguaje ensamblador.
  6. Ensamblar y Cargar el módulo ejecutable en memoria : `Execute → Assemble & Load`
  7. Analizar el mapa de memoria : sección de instrucciones y sección de datos
  8. Activar el modo depuración : `Execute → Debug Mode`
  9. Ejecución de cada instrucción paso a paso : `Step by Step`
- Contenido de la Memoria
  - La primera instrucción está almacenada en los 20 bits de la izda de la posición de memoria y la segunda instrucción en la dcha.

./images/von_neumann/ias_code_machine.png

Figura 6: IAS Codigo Maquina

- Contenido de los Registros:

./images/von_neumann/ias_registers.png

Figura 7: IAS Registros

- Ejercicio
  - Antes de la ejecución de cada instrucción interpretarla: interpretar la instrucción en lenguaje máquina.
  - preveer el nuevo contenido de la sección de datos de la memoria
  - preveer el nuevo contenido de los registros de la CPU.
  - preveer la próxima instrucción a ejecutar
  - Deducir el organigrama del programa.

8.5. Ejercicios

8.5.1. Tutorial2: Producto/Cociente

- Desarrollar el programa que realice la operación $N(N+1)/2$ equivalente a obtener el resultado de la suma del Tutorial1 $\sum_{i=1}^{N} i = N(N+1)/2$.
  - Pseudocódigo del algoritmo
  - Organigrama del algoritmo
  - Programa en lenguaje RTL→ Comentar apropiadamente el programa (cabecera con metainformación,secciones estructurales,bloques funcionales).
  - Programa en lenguaje Ensamblador
  - Ejecutar el programa paso a paso analizando el valor de los registros al ejecutar la multiplicación y la división.
- A TENER EN CUENTA en la descripción RTL:
  - El producto de dos números de M dígitos da como resultado un número de 2M dígitos, es decir, el doble que los multiplicandos. Esto dificulta las operaciones aritméticas posteriores a la multiplación en la expresión matemática. Por ello dejaremos la operación multiplicación para el final dando prioridad a la suma y a la división
    - $N(N+1)/2 = ((N+1)/2) * N$
  - La división puede tener resto 1 ó 0 dependiendo de si el dividendo es par o impar
    - Si N es impar → (N+1)/2 tiene el resto 0 → $((N+1)/2) * N$ donde N+1 es par
    - Si N es par → (N+1)/2 tiene el resto 1 → (N+1) = Cociente*2+Resto → $N(N+1)/2 = N*C+N/2$ donde N es par
  - La división por una potencia de 2 como 2^1 se realiza mediante una operación lógica: desplazar 1 bit a la izda el dividendo. El número de bits a desplazar es el valor del exponente.
    - descripción RTL **AC** ← **AC<<1**

8.5.2. Organigramas

Diagrama de Flujo: Alto Nivel y RTL

8.5.3. Tutorial3: Vectores

- Realizar la suma $C = A + B$ de dos vectores A y B de 10 elementos cada uno inicializados ambos con los valores del 1 al 10.

---

nota
Para acceder a cada elemento de un vector es necesario ir incrementando la dirección absoluta de memoria del operando en la instrucción que accede a los elementos del vector, por lo tanto, es necesario modificar el campo de operando de la instrucción. Hay una instrucción de transferencia de los 12 bits del campo de operando a los 12 bits de menor peso del registro AC , es decir,
AC(28:39) $\leftarrow$ M[operando](8:19) . Y otra instrucción que realiza la transferencia inversa M[operando (8:19)] $\leftarrow$ AC(28:39).
De esta manera se pueden realizar operaciones de aritméticas y lógicas sobre los 12 bits del campo de operando de una instrucción.

- Pseudocódigo del algoritmo
- Organigrama del algoritmo
- Programa en lenguaje RTL $\rightarrow$ Comentar apropiadamente el programa (cabecera con metainformación,secciones estructurales,bloques funcionales).
- Programa en lenguaje Ensamblador: Se aconseja no realizar el programa directamente en su totalidad sino por fases, comenzando por una versión sencilla e ir avanzando hasta completar el programa en la versión final. Por ejemplo:
  - 1ª versión : Inicializar el vector A[i]=i
  - 2ª versión : Inicializar los vectores A[i]=i, B[i]=i, C[i]=i
  - 3ª versión : C[i]=A[i]+B[i]
  - Posibles variables : len:longitud del vector, A0: dirección del primer elemento del Vector A, i: índice del vector, etc.
- Ejecutar el programa paso a paso depurando las distintas versiones del programa.

8.5.4. Organigramas (1ª versión)

Diagrama de Flujo: Alto Nivel y RTL

9. Operación de la Máquina IAS

Figura 8: IAS Operation

- Operación de la máquina IAS:
  - El ciclo de instrucción tiene dos FASES
  - La primera fase es común a todas las instrucciones.
- Ejemplos de instrucciones
  - X: referencia del operando
  - AC $\leftarrow$ M(X)
  - GOTO M(X,0:19): salto incondicional a la dirección X. X apunta a dos instrucciones. X,0:19 es la referencia de la Instrucción de la izda.
  - If AC>0 goto M(X,0:19): salto condicional
  - AC $\leftarrow$ AC+M(x).

10. Conclusiones

1. Para la programación de bajo nivel es necesario conocer las principales características de la arquitectura ISA de la computadora : Estructura de la computadora , Formato de datos e instrucciones y repertorio de instrucciones.
2. La programación en lenguaje ensamblador no se realiza directamente en dicho lenguaje sino que se sigue una estrategia top-down comenzando por una descripción en lenguje de pseudocódigo.
3. Es el diseño de repertorio de instrucciones ISA de la computadora el que facilita o dificulta la programación de bajo nivel. Un repertorio excesivamente limitado como la máquina IAS de von Neumann dificulta la realización de expresiones matemáticas tan sencillas como una multiplicación seguida de la división. La secuencia de instrucciones RTL deberá tener en cuenta el facilitar el desarrollo del algoritmo.
4. La programación del algoritmo en lenguaje ensamblador sigue una estrategia ascendente comenzando por una versión incompleta y lo más sencilla posible del programa a desarrollar.
5. Cada versión desarrollada del programa en lenguaje ensamblador ha de ser depurada y verificada mediante el simulador IASSim.

11. Ejercicios

11.1. Arquitectura von Neumann

1. You are to write an IAS program to compute the results of the following equation. and N are positive integers with N > 1. Note: The IAS did not have assembly language only machine language.

   a. Use the equation Sum(Y)= N(N+1)/2 when writing the IAS program.

   - Desarrollo:

```
; Suma de los primeros N numeros enteros. Y=N(N+1)/2
; CPU IAS
; lenguaje ensamblador: simaulador IASSim
; Ejercicio 2.1 del libro de William Stalling, Estructura de Computadores

; SECCION DE INSTRUCCIONES
S(x)->Ac+ n    ;01 n   ;AC    <- M[n]
S(x)->Ah+ uno  ;05 uno ;AC    <- AC+1
At->S(x) y     ;11 y   ;M[y]  <- AC
S(x)->R y      ;09 y   ;AR    <- M[y]
S(x)*R->A n    ;0B n   ;AC:AR <- AR*M[n]
R->A           ;0A     ;AC    <- AR
A/S(x)->R dos  ;0C 2   ;AR    <- AC/2
R->A           ;0A     ;AC    <- AR
At->S(x) y     ;11 y   ;M[y]  <- AC
halt
; como el número de instrucciones  es par no es necesaria la directiva .empty


; SECCION DE DATOS
; Declaracion e inicializacion de variables
y:     .data 0  ;resultado

; Declaracion de las Constantes
n:    .data 5   ;parametro N
uno:  .data 1
dos:  .data 2
```

   b. Do it the “hard way,” without using the equation from part (a).

   - Desarrollo:

```
; adds up the values n+...+3+2+1(+0) in a loop and stores
; the sum in memory at the location labeled "sum"

loop:   S(x)->Ac+  n    ;load n into AC
        Cc->S(x)   pos  ;if AC >= 0, jump to pos
        halt            ;otherwise done
        .empty          ;a 20-bit 0
pos:    S(x)->Ah+  sum  ;add n to the sum
        At->S(x)   sum  ;put total back at sum
        S(x)->Ac+  n    ;load n into AC
        S(x)->Ah-  one  ;decrement n
        At->S(x)   n    ;store decremented n
        Cu->S(x)   loop ;go back and do it again

n:     .data 5  ;will loop 6 times total
one:   .data 1  ;constant for decrementing n
sum:   .data 0  ;where the running/final total is kept
```

. a. On the IAS, what would the machine code instruction look like to load the contents of memory address 2 to the accumulator?

- Desarrollo: 0x01002

  a. How many trips to memory does the CPU need to make to complete this instruction during the instruction cycle?

  - Desarrollo: Dos accesos: captura de la instrucción y captura del operando

    1. On the IAS, describe in English the process that the CPU must undertake to read a value from memory and to write a value to memory in terms of what is put into the MAR, MBR, address bus, data bus, and control bus.

```
Lectura
MAR         <- address
Address Bus <- MAR
Control Bus <- Read
Data Bus    <- Data
MBR         <- Data Bus

Escritura
MAR         <- address
Data Bus    <- MBR
Control Bus <- Write
```

    2. Given the memory contents of the IAS computer shown below,

```
Address Contents
08A 010FA210FB
08B 010FA0F08D
08C 020FA210FB
```

- show the assembly language code for the program, starting at address 08A. Explain what this program does.

  - Desarrollo:

Address	Contents	RTL		Instructions	
08A	010FA210FB	AC←M[0FA]	M[0FB]←AC	LOAD M[0FA]	STORE M[0FB]
08B	010FA0F08D	AC←M[0FA]	AC>0:PC←0x08D(0:19)	LOAD M[0FA]	JMP +M[08D(0:19)]
08C	020FA210FB	AC←−M[0FA]	M[0FB]←AC	LOAD −M[0FA]	STORE M[0FB]

  - El programa realiza la siguiente función:

- Si el contenido de 0x0FA es positivo copia el contenido de memoria de la posición 0x0FA a la posición 0xFB y salta a la posición 0x08D, dejando en el acumulador el contenido de 0xFA.Si Si el contenido de 0x0FA es negativo copia el contenido de memoria de la posición 0x0FA a la posición 0xFB cambiado de y salta a la posición 0x08D, dejando en el acumulador el contenido de 0xFA en positivo, es decir, el módulo.

  1. In Figure 2.3, indicate the width, in bits, of each data path (e.g., between AC and ALU).

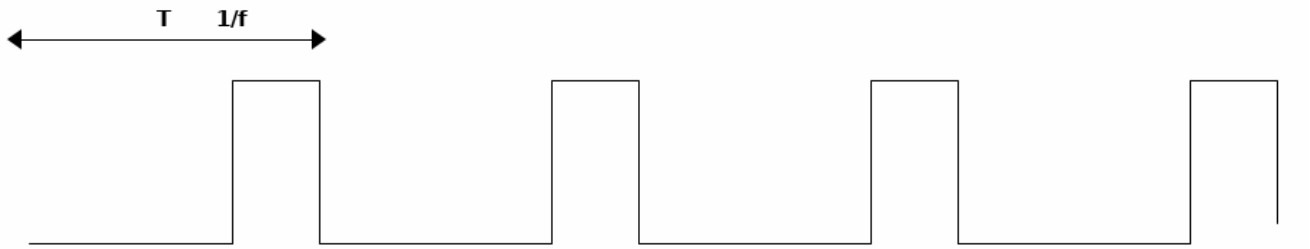
     - Desarrollo:

- AC, AR y MBR 40 bits
- IBR 20 bits
- MAR y PC 12 bits
- IR 8 bits

1. The ENIAC was a decimal machine, where a register was represented by a ring of 10 vacuum tubes. At any time, only one vacuum tube was in the ON state, representing one of the 10 digits. Assuming that ENIAC had the capability to have multiple vacuum tubes in the ON and OFF state simultaneously, why is this representation “wasteful” and what range of integer values could we represent using the 10 vacuum tubes?
   - Desarrollo: Con 10 tubos únicamente podemos representar los digitos 0-9
2. 2.10 A benchmark program is run on a 40 MHz processor. The executed program consists of 100,000 instruction executions, with the following instruction mix and clock cycle count:

Instruction_Type	Instruction_Count	Cycles_per_Instruction
Integer_arithmetic	45,000	1
data_transfer	32,000	2
Floating_point	15,000	2
Control_transfer	8000	2

- Determine the effective CPI, MIPS rate, and execution time for this program.
  - Desarrollo:
- Reloj de la CPU

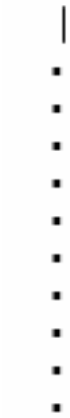


- T=1/f= 25ns: ciclo del reloj de la CPU: duración mínima de una microoperación.
- CPI: ciclos por instrucción: valor medio = 1*(45/100)+2*(32/100)+2*(15/100)+2*(8/100)=0,45+0,64+0,30+0,16=1.55
- MIPS: Millones de Inst. por seg: (1/CPI)(inst/ciclo)* F_clock(ciclos/seg)*10^{-6}=(1/1.55)*40*10^{6}*10^{-6}=25.8
- T=(1/MIPS)(seg/millones de instr)*100.000*10^{-6}=3.87ms

11.2. Interconexión CPU-Memoria

1. The hypothetical machine of Figure 3.4 also has two I/O instructions: 0011 Load AC from I/O 0011 Store AC to I/O
   - In these cases, the 12-bit address identifies a particular I/O device. Show the program execution (using the format of Figure 3.5) for the following program:
     1. Load AC from device 5.
     2. Add contents of memory location 940.
     3. Store AC to device 6.
   - Assume that the next value retrieved from device 5 is 3 and that location 940 contains a value of 2.
     - Desarrollo:

1. Consider a hypothetical microprocessor generating a 16-bit address (for example, assume that the program counter and the address registers are 16 bits wide) and having a 16-bit data bus.
    a. What is the maximum memory address space that the processor can access directly if it is connected to a “16-bit memory”?
    b. What is the maximum memory address space that the processor can access directly if it is connected to an “8-bit memory”?
    c. What architectural features will allow this microprocessor to access a separate “I/O space”?
    d. If an input and an output instruction can specify an 8-bit I/O port number, how many 8-bit I/O ports can the microprocessor support? How many 16-bit I/O ports? Explain.
        - Desarrollo:
            - Dibujar el esquema de buses que visualice:
                - interconexiones periféricos, puertos, controlador E/S, memoria principal, CPU, buses



            - Espacio de direcciones: conjunto de direcciones de un mismo bus de direcciones. La capacidad se expresa en BYTES.

- El Espacio Memoria Principal y el Espacio Controlador E/S son espacios diferentes. Comparten el mismo bus de direccione del bus del sistema pero hay una señal de control que activa la conexión con la memoria principal o con el controlador E/S.
- "16 bit memory": 16 bits word size → data bus
- "8 bit memory": 8 bits word size → data bus
- I/O port number: numero de puertos del controlador E/S. Cada puerto para un periférico. Se diferencia el puerto de entrada del puerto de salida. 8 bit I/O port es un puerto con un data buffer de 8 bits y un 16 bit I/O port es un puerto con un data buffer de 16 bits.
- a)2^{16}Bytes. 64KB. En el bus de datos se transfieren datos de dos bytes.
- b)2^{16}Bytes. 64KB. En el bus de datos se transfieren datos de un byte.
- c)En el bus del sistema hace falta una señal de control : señal I/O
- d) 2^8= 256 puertos de entrada y 256 puertos de salida en el controlador E/S independientemente del tamaño del buffer I/O si es de 8 bits o 16 bits.

2. Consider a 32-bit microprocessor, with a 16-bit external data bus, driven by an 8-MHz input clock. Assume that this microprocessor has a bus cycle whose minimum duration equals four input clock cycles. What is the maximum data transfer rate across the bus that this microprocessor can sustain, in bytes/s? To increase its performance, would it be better to make its external data bus 32 bits or to double the external clock frequency supplied to the microprocessor? State any other assumptions you make, and explain. Hint: Determine the number of bytes that can be transferred per bus cycle.

   - Desarrollo:
     - 32-bit CPU : tamaño de los registros internos de la CPU. Bus de datos local (interno) de la CPU
     - 16-bit external data bus: bus de datos del sistema
     - CPU input clock: 8MHz
     - bus cycle: ciclo del bus del sistema: duración 4 veces el de la CPU : 2 MHz.
     - a) Data transfer rate: teóricamente número de datos en la secuencia continua de una transferencia cada *bus cycle* durante 1 segundo: 2MTransferencias/s. Cada transferencia el bus de datos transfiere 16 bits, es decir, 2 bytes = 2M/s*2B = 4MB/s
     - b) Doblar el ancho del bus de datos, dobla el ancho de banda → 8MB/s
     - c) Doblar la frecuencia de reloj reduce proporcionalmente el ciclo de bus y dobla el ancho de banda → 8MB/s

3. Consider two microprocessors having 8- and 16-bit-wide external data buses, respectively. The two processors are identical otherwise and their bus cycles take just as long.

   a. Suppose all instructions and operands are two bytes long. By what factor do the maximum data transfer rates differ?

   b. Repeat assuming that half of the operands and instructions are one byte long.

      - Desarrollo:
        - a) CPU1 de 8 bits tiene un ancho de banda mitad (50 %) respecto de CPU2 de 16 bits
        - b1) CPU1: 50 % de 2 bytes a 2 ciclos de bus por cada 2 bytes y el otro 50 % de 1 byte en 1 ciclo por byte =2ciclos*50 %+1ciclo*50 %=1.5ciclos
        - b2) CPU2: 50 % de 2 bytes a 1 ciclo de bus por cada 2 bytes y el otro 50 % de 1 byte a 1 ciclo de bus (unicamente se puede acceder a una instrucción o un dato en cada ciclo de bus)=1ciclo*50 %+1ciclos*50 %=0.5+0.5=1ciclos
        - b) según b1 y b2 la CPU1 tiene un ancho de banda 150 % menor que la CPU2, es decir, el 66.6 % del CPU2.

4. A microprocessor has an increment memory direct instruction, which adds 1 to the value in a memory location. The instruction has five stages: fetch opcode (four bus clock cycles), fetch operand address (three cycles), fetch operand (three cycles), add 1 to operand (three cycles), and store operand (three cycles).

   a. By what amount (in percent) will the duration of the instruction increase if we have to insert two bus wait states in each memory read and memory write operation?

   b. Repeat assuming that the increment operation takes 13 cycles instead of 3 cycles.

      - Desarrollo:
        - instruction cycle: 4+3+3+3+3= 16ciclos

- a) accesos a memoria en las 3 etapas fetch y en la etapa store → incremento de 2*4 ciclos de espera → incremento de 8/16 → un incremento del 50 %
- b) instruction cycle:4+3+3+13+3= 26ciclos → incremento del ciclo de instruccion en un 8/26 → incrmento en 34 %

5. The Intel 8088 microprocessor has a read bus timing similar to that of Figure 3.19, but requires four processor clock cycles. The valid data is on the bus for an amount of time that extends into the fourth processor clock cycle. Assume a processor clock rate of 8 MHz.

   a. What is the maximum data transfer rate?

   b. Repeat but assume the need to insert one wait state per byte transferred.

   - Desarrollo:
     - 8088: bus data: 1 byte
     - read time : 4 cpu cycles
     - data valid: 1 processor clock cycle. El cuarto ciclo del read time.
     - cpu clock: 8MHz
     - a) 4 ciclos por transferencia. 8MHz/4ciclos = 2MT/s = 1 byte por transferencia → 2MB/s
     - b) cada transferencia está un ciclo sin transferir (4,1) → throughput = 4/5 del máximo → (4/5)*2MB/s→1.6MB/s

6. The Intel 8086 is a 16-bit processor similar in many ways to the 8-bit 8088. The 8086 uses a 16-bit bus that can transfer 2 bytes at a time, provided that the lower-order byte has an even address. However, the 8086 allows both even- and odd-aligned word operands. If an odd-aligned word is referenced, two memory cycles, each consisting of four bus cycles, are required to transfer the word. Consider an instruction on the 8086 that involves two 16-bit operands. How long does it take to fetch the operands? Give the range of possible answers. Assume a clocking rate of 4 MHz and no wait states

   - Desarrollo:
     - 8086: bus data: 2 bytes
     - intel : little endian: el LSB byte se guarda en la dirección menor y el MSB byte en la dirección superior.
     - alineación del dato par requiere 1 ciclo de memoria.
     - palabras con alineación impar requieren 2 ciclos de memoria. Cada ciclo de memoria son 4 ciclos de bus.
     - instrucción de 2 operandos de 2 bytes cada uno. CPU clock de 4MHz → 0.250 microsegundos → 250 ns
     - a) los dos operandos tienen alineación par
       - 1 ciclo de memoria cada operando: 2 ciclos de memoria: 8 ciclos de bus → 2 microsegundos
     - b) un operando tiene alineación par y el otro impar
       - 1 ciclo de memoria el par y 2 ciclos el impar: 3 ciclos de memoria: 12 ciclos de bus → 3 microsegundos
     - c) los dos operandos tienen alineación impar
       - 2 ciclos cada operando: 4 ciclos de memoria: 16 ciclos de bus → 4 microsegundos.

7. Consider a 32-bit microprocessor whose bus cycle is the same duration as that of a 16-bit microprocessor. Assume that, on average, 20 % of the operands and instructions are 32 bits long, 40 % are 16 bits long, and 40 % are only 8 bits long. Calculate the improvement achieved when fetching instructions and operands with the 32-bit microprocessor.

   - Desarrollo
     - En cada flanco positivo del ciclo de bus se realiza una transferencia entre memoria y CPU. La cpu de 16 bits realiza una transferencia de 2 bytes o menos y el de 32 bits una transferencia de 4 bytes o menos.
     - Media ciclos (CPU 16 bits)= 0.2x(2ciclos para las dos transferencias de 2 bytes cada una)+0.4x1+0.4x1=1.2 ciclos de media
     - Media ciclos (CPU 32 bits)= 0.2x1+0.4x1+0.4x1=1ciclo de media
     - Mejora de (1.2-1) sobre 1.2 = (1.2-1)/1.2=17 %