Ejercicios de los Temas 1-9

HISTORIAL DE REVISIONES

NÚMERO	FECHA	MODIFICACIONES	NOMBRE
0.6.0	2018 September 3		C

Índice

1. Arquitectura von Neumann

1.1. Computadoras: IAS, ENIAC, ..

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
```

2. On the IAS, what would the machine code instruction look like to load the contents of memory address 2 to the accumulator? How many trips to memory does the CPU need to make to complete this instruction during the instruction cycle?

   - Desarrollo:
     - 0x01002
     - Dos accesos: captura de la instrucción y captura del operando

3. 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
```

4. 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.

5. Indicate the width, in bits, of each data path (e.g., between AC and ALU) of IAS microarchitecture.

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

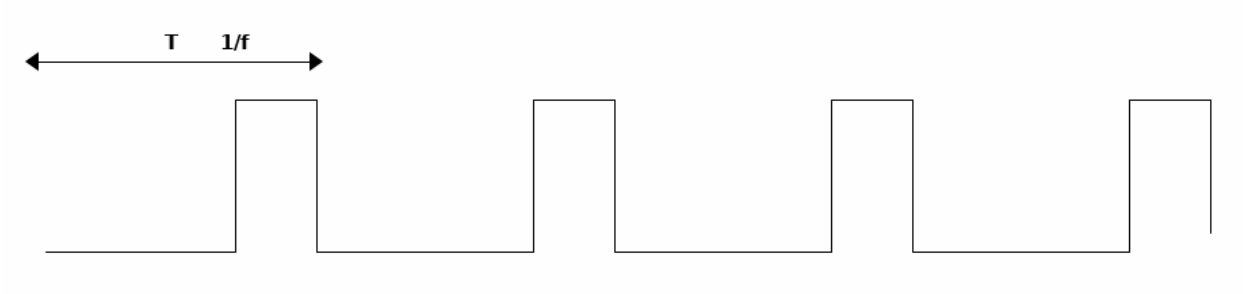
6. The ENIAC was a decimal machine, where a register was represented by a ring of 10 vacuum tubes. At any time, only one vacuum tube was in the ON state, representing one of the 10 digits. Assuming that ENIAC had the capability to have multiple vacuum tubes in the ON and OFF state simultaneously, why 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

7. 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

1.2. Interconexión CPU-Memoria

1. The hypothetical machine 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 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:

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

2. 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.

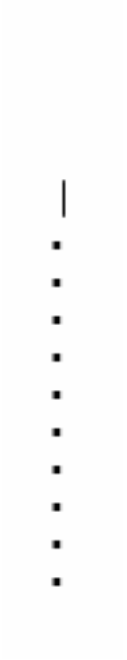
   a. What is the maximum memory address space that the processor can access directly if it is connected to a “16-bit memory”?

   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.

3. 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

4. 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.

5. 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 → incremento en 34 %

6. 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

7. 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.

8. 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 %

2. Representación de Datos

1. Representar el número decimal 1197 en las bases:
   a. Hexadecimal
   b. octal:
   c. binaria:
   d. Representar el número 0x4AD en base binaria y base octal mediante una conversión directa, sin calcular su valor.
2. Representar el número -1197 en formato:
   a. Signo-magnitud:
   b. Complemento a 2:
3. Calcular el rango de los números enteros de 8 bits en complemento a 2.
4. Utilizar notación hexadecimal:
   a. Representar el valor 23 en el formato BCD..
      - En el formato Binary Code Decimal (BCD) cada dígito decimal se expande independientemente en su código binario de 4 bits
      - 2→0010 ; 3→0011 ; 23 → 0010-0011
   b. The ASCII characters 23
      - 0x32-0x33 → 0011-0010-0011-0011
5. For each of the following packed decimal numbers, show the decimal value:
   a. 0111 0011 0000 1001
      - 7309
   b. 0101 1000 0010
      - 582
   c. 0100 1010 0110
      - No es posible ya que 1010 corresponde al valor 10 que no tiene un digito decimal sino dos.
6. Another representation of binary integers that is sometimes encountered is ones complement. Positive integers are represented in the same way as sign magnitude.A negative integer is represented by taking the Boolean complement of each bit of the corresponding positive number. Note: Ones complement arithmetic disappeared from hardware in the 1960s, but still survives checksum calculations for the Internet Protocol (IP) and the Transmission Control Protocol (TCP).
   a. Provide a definition of ones complement numbers using a weighted sum of bits.
      - Poner ejemplos de conversión con n=3 bits
        - 000→111 (luego el cero tiene dos representaciones), +*1*: 001→ -*1*:110, +*2*:010→'-2':101, +*1*:011→'-3':100
      - positivos con n bits→ $\sum_{i=0}^{n-1} b_i 2^i$.
      - negativos con n bits

- Tenemos en cuenta que complemento a dos = complemento_a_1 + 1
- el complemento a dos con n bits de X se puede calcular como la resta binaria 2^n (en binario) - X : por ejemplo con 3 bits el complemento a dos de *+1* es 1000-1=111
- el complemento a 1 es el complemento a 2 menos 1 $\rightarrow$ 2^n -X -1. Por ejemplo con 3 bits el complemento a uno de *+1* 1000-1-1 = 110

b. What is the range of numbers that can be represented in ones complement with n bits?

- El máximo positivo $\rightarrow$ *011__1* : 2^n-1
- El máximo negativo $\rightarrow$ *100__0* : -(2^n-1)

7. Representar 0.56789 en binario utilizando multiplicaciones sucesivas

```
0.56789 * 2 = 1.13578 = 1 + 0.13578 -> 1, bit de la posición -1
0.13578 * 2 = 0.27156 -> 0, bit de la posición -2
0.27156 * 2 = 0.54312 -> 0, bit de la posición -3
0.54312 * 2 = 1.08624 = 1 + 0.08624 -> 1, bit de la posición -4
```

8. Representar 0.0625 en binario sin utilizar multiplicaciones sucesivas.

- 0.0625 = M*2^E tal que E es un entero
- log_dos(0.0625) = log_dos(M)+E
- -4 = log_dos(M)+E $\rightarrow$ E=-4 y log_dos(M)=0 $\rightarrow$ M=1

9. Representar el número real 1234.56789 en base binaria:

- En formato coma fija

```
Parte Entera: 1234 : 10011010010
Parte Fracción: 0.6789: 0.10010001011000010
Número 12345.6789: 10011010010.10010001011000010
```

- En notación científica: 1.00110100101001000101100001 0*2+10
- En precisión simple punto flotante:

```
Campo signo: + : 0
Campo Exponente (8 bits): 10+127 = 137 = 10001001
Campo fracción mantisa (23 bits)= 00110100101001000101100
```

- En precisión doble punto flotante:

```
Campo signo: + : 0
Campo Exponente (11 bits): 10+1023 = 1033 = 10000001001
Campo fracción mantisa (52 bits) = 0011010010100100010110000010__0
```

10. Codificar el número entero 3 en single precision FP

- 3 = 11 = 1.1 2^1
  - S=0, E=1+127=128, Mn=0.1
  - 0-1000-0000-1000-0000-ceros
  - No es necesario redondear
  - Resultado= 0x40400000

11. Representar el número natural 123456789 en precisión simple Punto Flotante (IEEE-754)

- 123456789 = 0x075BCD15 = 111-0101-1011-1100-1101-0001-0101 = 1.11010110111100110100010101*2^{+26}
- Redondear= 1.11010110111100110100011*2^{+26}
  - Campo Signo= 0

- Campo Exp= 26+127=153=10011001
- Campo fracción Mantisa=0.11010110111100110100011
- Resultado 0x4CEB79A3

12. Float Point:

    a. Consider a fixed-point representation using decimal digits, in which the implied radix point can be in any position (e.g., to the right of the least significant digit, to the right of the most significant digit, and so on). How many decimal digits are needed to represent the approximations of both Planck's (6.63×10^{-27}) constant and Avogadro's number (6.02×10^{23}) The implied radix point must be in the same position for both numbers.

    - para el número de planck hace falta correr la coma 27 posiciones a la izda más los dos digitos a la derecha (63) → fracción de 27 dígitos
    - para el número de avogrado hace falta correr la coma 23 posiciones a la dcha más el dígito de la izda (7) → parte entera de 24 dígitos
    - para los dos 29+24=53 dígitos

    b. Now consider a decimal floating-point format with the exponent stored in a biased representation with a bias of 50. A normalized representation is assumed. How many decimal digits are needed to represent these constants in this floating point format?

    - planck → $0.63\times 10^{-26} \rightarrow 0.63\text{x } 10^{-26+50} \rightarrow 0.63\text{x } 10^{+24}$
    - avogrado → $0.602 \text{ x } 10^{24} \rightarrow 0.602 \text{ x } 10^{24+50} \rightarrow 0.602 \text{ x } 10^{74}$
    - para los dos hacen falta = 3 dígitos fraccíon y 2 dígitos para el exponente.

13. Any floating-point representation used in a computer can represent only certain real numbers exactly; all others must be approximated. If Ap is the stored value approximating the real value A, then the relative error, r, is expressed as r=(A-Ap)/A .Represent the decimal quantity *+0.4* in the following floating-point format: base=2 exponent: biased, 4 bits; significand, 7 bits.What is the relative error?

    - En binario coma fija
        - 0.4x2=0.8 →0
        - 0.8x2=1.6 →1
        - 0.6x2=1.2 →1
        - 0.2x2=0.4 →0
        - 0.4x2 → otra vez 0110, luego es un número periódico → 0.0110-0110-0110-período
        - normalizado 1.10-0110-0110-etc x 2^{-1} → se representa la fracción 10-0110-0110-
        - con fracción de 7 bits 1001100
        - Valor del número aproximado $1.1001100 \text{ x } 2^{-1} = 110011 \text{ x } 2^{-6} = (32+16+2+1)\text{x}2^{-6}=51/64=.796875$
        - Error=(0.8-0.796875)/0.8=.003906250=0.4 %
        - En exceso de 4 bits. Con el número excedido el rango es (0,15). El exceso es la mitad de combinaciones -1=16/2-1=7, luego se pueden representar los exponentes (-7,8).

14. Representar el número Pi en coma flotante IEEE de simple y doble precisión

    - SOLUCION:
    - Formato decimal: 3.14159265358979323846264338327950288419 68
    - Binario Coma fija : 11.00100100001111111011010101000100010000101110100011000010001101001100001001100011001100
    - Hexadecimal Coma fija: 0011.0010-0100-0011-1111-0110-1010-1000-1000-1000-0101-1010-0011-0000-1000-1101-0011-0001-0011-0001-1001-1000-1010-0010-1110-0000-0011-0111
        - 3.243F6A8885A308D313198A2E03E

Figura 1: IEEE-754 precisión simple (32 bits)

- v = s × 2^e × m
  - Notación científica con la mantisa normalizada y su parte fracción truncada a 23 bits y redondeada:
  - *+ (1 + 0.10010010000111111011011) * 2^{+1}*
- Campos:
  - Signo : positivo --→ 0 --→ 1 bit
  - Exponente:+1
  - Exponente desplazado + *127 = +1+127 = 128 → 10000000 → 8 bits*
  - Mantisa normalizada:1 + 0.10010010000111111011011
    - Fracción de la mantisa normalizada: 10010010000111111011011 → 23 bits

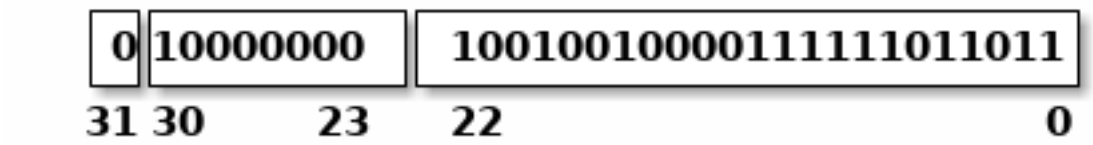


Figura 2: Formato IEEE-754 precisión simple

- Resultado= 0x40490fdb

3. Operaciones Aritméticas

1. Sumar en binario puro
   - 10011011+00011011 SOLUCION: = 10110110
   - 0x3A1F+0xF4E1 SOLUCION: = 0x12f00
   - 10011011+10011011 SOLUCION: = 100110110
2. Sumar en complemento a 2 : 50+23
   - Realizar las operaciones en código binario SOLUCION: = 0110010+0010111 = 01001001
   - Realizar las operaciones en código hexadecimal SOLUCION: = 0x32+0x17 = 0x49
3. Representar el valor -66 en complemento a 2
   - SOLUCION: *+66* = 01000010 $\rightarrow$ -66 = 10111101+1 = 10111110
4. Resta en complemento a 2 : 33-66
   - Realizar las operaciones en código binario: SOLUCION: 0100001+10111110=1011111
   - Realizar las operaciones en código hexadecimal: SOLUCION: 0x21+0xBE=0xDF
5. Representar en hexadecimal el mayor número en módulo que se puede representar en complemento a 2 con 16 bits
   - SOLUCION: 1000-0000-0000-0000 = 0x8000
6. Sumar en complemento a 2 con 16 bits 0x8000+0x8000
   - SOLUCION: 0x8000+0x8000=0x0000
7. Restar en binario puro 0110010 - 0010111
   - SOLUCION:

```
 0110010  <- minuendo
 0010111  <- sustraendo
   1111   <- llevadas
 *******
 0011011
```

8. Restar en hexadecimal 0x32-0x17

```
  0x32  <- minuendo
  0x17  <- sustraendo
    1   <- llevadas
*****
 0x1B
```

9. Multiplicación
   - ¿A qué equivale en base binaria multiplicar por una potencia de 2 positiva 2^n? :
     - SOLUCION:
     - en un número real mover la coma n posiciones hacia la dcha
     - en un número entero añadir n ceros a la dcha
     - en un registro es desplazar los bits n posiciones hacia la izda e introducir n ceros por la dcha
   - ¿A qué equivale en base binaria multiplicar por una potencia de 2 negativa 2^{-n}? :
     - SOLUCION:

- en un número real mover la coma n posiciones hacia la izda
- en un número entero añadir n ceros a la izda
- en un registro es desplazar los bits n posiciones hacia la dcha e introducir n ceros por la izda

10. Realizar la multiplicación de los siguientes números naturales:

- 1010*1010
- 1010*1111
- SOLUCION:

```
              1010            1010
            x 1010          x 1111
           ********        ********
              0000            1010
             1010            1010
            0000            1010
           1010            1010
           ********        ********
           1100100        10010110
```

4. Operaciones Lógicas

1. Realizar las operaciones lógicas Ã, Ã+1, A+B, A·B, A⊕B siendo A=10101010 y B=11110000

   - SOLUCION:
   - A+B → A|B → AVB
   - A·B → A&B → A∧B
   - Ã = 01010101
   - Ã+1 = 01010110
   - A+B = 11111010
   - A*B = 10100000
   - A⊕B = 01011010

2. Dado un operando de 20 bits, indicar la operación lógica a realizar para: (expresar la operación con los operandos en código hexadecimal)

   - Set el bit 7 (posición 7ª)
     - SOLUCION: 20 bits son 5 dígitos hex → Operando | 0x00080
   - Clear el bit 15
     - SOLUCION: Operando & 0xF7FFF
   - Toogle el bit 19
     - SOLUCION:Operando ⊕ 0x8000
   - Set toda la palabra
     - SOLUCION: Operando | 0xFFFF
   - Clear toda la palabra
     - SOLUCION:Operando ⊕ Operando ó Operando & 0x00000

3. Dadas las operaciones lógicas SAR X,n(shift arithmetic right X, n), SLR X,n(shift logic right X,n) , SAL X,n(shift arithmetic left X, n) y SLL X,n(shift logic left X, n) donde X es el operando, xxR significa derecha, xxL significa izquierda y n es el número de posiciones a desplazar. Realizar las siguientes operaciones con el operando A=10101010: SAR A,4, SLR A,4, SAL A, 4 y SLL A,4 de forma manual y también mediante un programa en lenguaje C:

   - SAR A,4 = 11111010
   - SLR A,4 = 00001010
   - SAL A,4 = 10101111
   - SLL A,4 = 10100000

4. Realizar la multiplicación $A \cdot 2^2$ y $A \cdot 2^{-2}$ donde A=10101010 primero manualmente y después mediante operaciones lógicas.

   - $A \cdot 2^2$ = 1010101000
   - $A \cdot 2^{-2}$ = 101010.10
   - Con operaciones lógicas:
     I. Doblar el tamaño de A → D:A ← 0000000001010101
     II. desplazar SLL D:A,2 → D:A ← 0000000101010100

5. Representación de las Instrucciones

1. Sea un computador con palabras de 32 bits. La CPU tiene 64 instrucciones diferentes de un operando, 32 registros de propósito general de 32 bits y posibilidad de direccionamiento directo a registro o indirecto con desplazamiento a registro-base.a)Diseñar el formato de instrucción para este computador. Debe especificar un registro de dirección y un desplazamiento, además del modo de direccionamiento y código de operación.b)¿Cuál es el máximo valor del desplazamiento (el desplaz. es un número en C2)?

   - SOLUCION

     a. Formato

     Palabra de 32 bits → Registros de propósito general de 32 bits.

     Formato de instrucciones con una estructura en 4 campos: código de operación, modo de direccionamiento, campo d operando (registro o registro con desplazamiento)

     1º Campo: código de operación: 64 instrucciones : $2^6 \rightarrow$ 6 bits

     2º Campo: modo de direccionamiento: 2 tipos: $2^1 \rightarrow$ 1 bit

     3º Campo: registro: 32 registros: $2^5 \rightarrow$ 5 bits

     4º Campo: Desplazamiento nº entero: (32-(6+1+5))bits → 20 bits

     b. Desplazamiento de 20 bits

     - En complemento a 2: Positivo máximo 0111-1111-1111-1111-1111 de valor 2^{19}-1 y Negativo mínimo 1000-0000-0000-0000-0000 que cambiado de signo es el 0-1000-0000-0000-0000-0000 de valor $+2^{19}$, por lo que el rango es $[+2^{19}-1,-2^{19}] \rightarrow$ [+524287,-524288]

2. Un computador con palabras de 24 bits posee 16 instrucciones diferentes de un operando, 8 registros para de propósito general, y 3 modos de direccionamiento (directo a registro, indirecto con registro e indirecto con desplazamiento a registro-base)

   a. Diseñar un formato de instrucción para este computador. Debe especificar el código de operación, el modo de direccionamiento, un registro y un desplazamiento.

   - SOLUCION:

     Word Size = 24 → Registros de Propósito general de 24 bits

     4 campos en el formato de instrucción:

     código de operación- modo de direccionamiento - Registro - Desplazamiento

     1º Campo: código de operación: 16 instrucciones : $2^4 \rightarrow$ 4 bits

     2º Campo: modo de direccionamiento: 3 tipos: $2^2 \rightarrow$ 2 bit

     3º Campo: registro: 8 registros: $2^3 \rightarrow$ 3 bits

     4º Campo: Desplazamiento nº entero: (24-(4+2+3))bits → 15 bits

   b. ¿Cuál es el rango de valores del desplazamiento en magnitud?, ¿y en C2?

   - Formato Magnitud: Mínimo el cero y el máximo 111-1111-1111-1111 = 2^{15}-1 = 32768
   - Complemento a 2: Positivo máximo 0111-1111-1111-1111-1111 de valor 2^{14}-1 y Negativo mínimo 100-0000-0000-0000 que cambiado de signo es el 0100-0000-0000-0000 de valor $+2^{14}$, por lo que el rango es $[+2^{14}-1,-2^{14}] \rightarrow$ [+16383,-16384]

3. Un computador tiene un formato de instrucción de 11 bits donde el campo de operando es de 4 bits. ¿Es posible codificar en este formato 5 instrucciones de dos operandos, 45 de un operando y 48 sin operando?. Justificar la respuesta.

   - SOLUCION
   - 3 tipos de formatos

- Tipo 1: campo tipo - Cod. Op. - Op1 - Op2
  - 2 bits - x bits - 4 bits - 4 bits → x=11-(2+4+4)= 1 bit → Máximo de 2 instrucciones < 5 instrucciones → No es posible
- Tipo 2: campo tipo - Cod. Op. - Op
  - 2 bits - y bits - 4 bits → y=11-(2+4)= 5 bits → Máximo de 32 instrucciones < 45 instrucciones → No es posible
- Tipo 3: campo tipo - Cod. Op.
  - 2 bits - z bits → 48 instrucciones

- Alternativa
  - Sin campo de tipo : 5+45+48=98 instrucciones → 2^7 → 7 bits → la instrucción tipo 1 ocuparía 7+4+4=15 bits > 11 → No es posible

4. Un computador de 16 bits de ancho de palabra (instrucciones, palabra de memoria, registros) y 8 registros, tiene el siguiente repertorio de instrucciones:

   - 14 instrucciones de referencia de un solo operando en memoria, con direccionamiento directo e indirecto de memoria
   - 31 instrucciones con dos operandos con los modos de direccionamiento directo e indirecto de registro.
   - 32 instrucciones sin operando explícito.

   a. Especificar la codificación de las instrucciones.
   b. Especificar la zona de memoria alcanzable en cada tipo de direccionamiento y rango posible de valores de los operandos (en C'2).

   - SOLUCION
     - Word Size = 16 → Registros de Propósito general de 16 bits
     - Repertorio con 3 tipos de formatos
     - 1º Tipo: Tipo-Cod.Op.-Modo Direc-Op1 → 14 instrucciones (2^4), Bits:2-4-1-x → x=16-(2+4+1)=9bits
     - 2º Tipo: Tipo-Cod.Op.-Modo Direc1-Op1-Modo Direc2-Op2 → 31 instrucciones (2^5) Bits:2-5-1-x-1-x → x=(16-(2+5+2))/2=3bits
     - 3º Tipo: Tipo-Cod.Op. → 32 instrucciones (2^5) Bits:2-5 → 7 bits ocupados de los 16.

5. Un computador basado en el 68000 presenta los siguientes contenidos en registro y memoria:

REGISTROS		**MEMORIA**	
Registro	Contenido	Dirección	Contenido
A1	100	99	104
A2	2	100	108
		101	106
		102	107
...	...	...	...
		199	100
		200	34
		201	96
		202	201

   - Si el contenido del desplazamiento de la instrucción en ejecución es desp=99 ¿Cuál seria el valor del operando (de tamaño byte) con los siguientes modos de direccionamientos?.

   I. Directo de memoria o absoluto (dirección = desplazamiento).
   II. Directo de registro con A1.
   III. Indirecto de registro con A1.
   IV. Indirecto con desplazamiento con registro base A1
   V. Indirecto con desplazamiento con registro base A2.
   VI. Indirecto con desplazamiento con registro base A1 e indexado con A2.
   VII. Indirecto de registro con predecremento con A1.

- SOLUCION:

```
Directo de memoria o absoluto (dirección = desplazamiento) -> M[99]=104
Directo de registro con A1. -> R[A1]=100
Indirecto de registro con A1. -> M[A1]=M[100]=108
Indirecto con desplazamiento con registro base A1 -> M[A1+99]=M[199]=100
Indirecto con desplazamiento con registro base A2.-> M[A2+99]=M[101]=106
Indirecto con desplazamiento con registro base A1 e indexado con A2.-> M[A1+99+A2]=M ↩
    [100+99+2]=M[201]=96
Indirecto de registro con predecremento con A1.-> M[A1-1]=M[100-1]=104
```

6. Un computador presenta los siguientes contenidos en registro y memoria:

REGISTROS		**MEMORIA**	
Registro	Contenido	Dirección	Contenido
R1	99	96	100
R2	6	97	102
		98	101
		99	104
		100	108
		101	106
		102	107
		103	109
		104	110

   - Si el contenido del desplazamiento de la instrucción en ejecución es 96 ¿Cuál seria el valor del operando con los siguientes direccionamientos?. a)Directo de memoria (dir = desp). b)Indirecto de memoria memoria (dir memoria = desp). c)Directo de registro con R1. d)Indirecto de registro con R1. e)Indirecto con desplazamiento con registro base R2

7. Se tiene un computador con un ancho de palabra de 32 bits y con un banco de registros de 32 registros de 32 bits. El computador tiene 64 instrucciones diferentes y los siguientes modos de direccionamiento: directo de memoria, indirecto de memoria e indirecto con desplazamiento a registro-base.

   a. Diseñar los dos formatos de las instrucciones de dos operandos sabiendo que siempre un operando está en memoria y otro en registro.
      - SOLUCION:
        - CodOp/Modo-Etiqueta_fuente/Registro_destino
          - CodOp=2^6=64 instrucciones
          - Modo: directo o indirecto:2^1
          - Etiqueta=2^5=32 bits de direcciones
          - Reg=2^5=32 bits
          - Total=6+1+5+5=17 bits
        - CodOp/Desplazamiento-Registro_fuente/Registro_destino
          - CodOp=2^6=64 instrucciones
          - Desplazamiento=2^5=32 bits de direcciones
          - Reg=2^5=32 bits
          - Total=6+5+5+5=21 bits
   b. Si cada dirección de memoria especifica un byte ¿qué zona de memoria se puede acceder con cada uno de los modos de direccionamiento?
      - SOLUCION

8. Consideremos cuatro arquitectura de procesador: acumulador, pila, memoria-memoria y registro-registro con 16 registros. Para las cuatro arquitectura se tienen los siguientes datos comunes:

   - El código de operación es siempre 1 byte

- Todas las direcciones de memoria son 2 bytes
- Todos los datos son 4 bytes
- Todas las instrucciones tienen una longitud igual a un numero entero de bytes

a. Escribir de forma genérica los programas en lenguaje ensamblador de cada una de las arquitecturas para realizar la siguiente operación; A=B+C. Para cada programa, calcular el tráfico con memoria y el tamaño del código. ¿Cuál es mas eficiente?.

b. Escribir los cuatro programas ensamblador para la siguiente secuencia de operaciones A=B+C ; B=A+C ; D=A-B Calcular el trafico con memoria y el tamaño del código. ¿Cuál es más eficiente?.

9. Considerando que en un procesador cada búsqueda de instrucción y cada acceso a un operando consumen un ciclo y teniendo en cuenta los siguientes datos en millones de referencias.

	TEX	**Spice**	**C**
Arquitectura R-R			
Referencias de datos	5.4	4.9	1.4
Palabras de instr.	14	18.9	3.9
Arquitectura M-M			
Referencias a datos	12.4	10.5	4.1
Palabras de instr	7.5	8.4	2.4

a. Calcular el porcentajes de accesos a memoria que son para buscar instruccio nes de los tres programas para la arquitectura R-R y para la arquitectura M-M.

b. ¿Cuál es la relación de accesos totales entre ambas arquitecturas?

10. Para la arquitectura M68000 de Motorola de 32 bits, mostrar el contenido de todos los registros y posiciones de memoria afectadas (sin incluir el PC) por la ejecución de cada una de las instrucciones, suponiendo que partimos siempre de las condiciones iniciales especificadas:

Instrucciones:	
a. CLR.L	-(A1)
b. CLR.W	D2
c. MOVE.W	$1204,D1
d. MOVE.W	#$1204,D1
e. MOVE.B	(A2)+,$1200
f. MOVE.L	D1,-(A2)
g. MOVE.L	(A1)+,D2

Condiciones iniciales:	
REGISTROS	MEMORIA
A1:00001202	0011FE:7777
A2:00001204	001200:1111
D1:01020304	001202:2222
D2:F0F1F2F3	001204:3388
	001206:4444
	001208:5555
	00120A:6666

- Sufijos → Long=4 bytes, Word=2Bytes, Byte=1Byte
- El incremento o decremento de la dirección efectiva se escala con el tamaño del operando
- SOLUCION

```
CLR.L -(A1) :
        Clear operando long
        Predecremento del registro A1 seguido de indirección
```

```
        A1<-A1-4 ;(A1-4=0x1202-0x4=0x11FE) A1:000011FE
        M[A1]<-0,M[A1+1]<-0,M[A1+2]<-0,M[A1+3]<-0  M[0011FE]:0000 M[001200]:0000
CLR.W D2 :
        Clear operando Word
        D2(15:0)<-0 ; D2:F0F10000
MOVE.W  $1204,D1
        Copiar 2bytes de Op_fuente (Dir. Directo) en Op_destino (Registro)
        D1(15:0)<-M[0x1204] ;(M[0x1204]=3388); D1:01023388
MOVE.W  #$1204,D1
        Copiar 2bytes de Op_fuente (Dir. Inmediato) en Op_destino (Registro)
        D1(15:0)<-0x1204 ; (D1:01021204)
MOVE.B  (A2)+,$1200
        Copiar 1byte Op_fuente (indirecto con postincremento), Op_destino (Directo)
        0x1200<-M[A2][LSB] ;(M[A2][LSB]=M[A2+1]=M[1205]=88) ; M[1200]:1188
        A2<-A2+1 ;(A2+1=0x1204+0x1=0x1205) ; A2:00001205
MOVE.L  D1,-(A2)
        Copiar 4bytes Op_fuente(Registro) a Op_destino(indirecto con predecremento)
        A2<-A2-4 ;(A2-4=0x1204-0x4=0x1200) A2:00001200
        M[A2+3]<-D1(7:0),M[A2+2]<-D1(15:8),M[A2+1]<-D1(23:16),M[A2]<-D1(31:24)  ; M ↩
            [001200]:01020304
MOVE.L  (A1)+,D2
        Copiar 4bytes Op_fuente(Indirecto con postincremento) a Op_destino(Registro)
        D2(7:0)<-M[A1+3] ; D2(15:8)<-M[A1+2]; D2(23:16)<-M[A1+1]; D2(31:24)<-M[A1] ;  ↩
            D2:11112222
        A1<-A1+4; (A1=0x1202+0x4=0x1206) A1:00001206
```

11. Mostrar el contenido de todos los registros y posiciones de memoria afectadas (sin incluir el PC) por la ejecución de cada una de las instrucciones, suponiendo que partimos siempre de las condiciones iniciales especificadas:

Instrucciones:	
a. MOVE.W -(A1),A3	
b. CLR.B -11(A2)	
c. MOVE.W (A4)+,-100(A1,D5.W)	
d. MOVE.W #$1FF,D5	

Condiciones iniciales:	
REGISTROS	MEMORIA
A1:00001504	001500:1234
A2:00001510	001502:5678
A3:11122233	001504:9ABC
A4:00001506	001506:EF11
D5:FA000064	001508:2233
	00150A:4455

12. Para la arquitectura M68000-32 de Motorola, suponiendo que se dan las siguientes condiciones iniciales, mostrar el contenido de todos los registros y posiciones de memoria afectadas (incluyendo el PC) por la ejecución de cada una de las instrucciones. Suponer, además, que las instrucciones están en posiciones consecutivas de memoria, a partir de la dirección $2000, y que se ejecutan en secuencia.

Condiciones iniciales:	
REGISTROS	MEMORIA
A1:00001504	001500:1234
A2:00001510	001502:5678
A4:00001506	001504:9ABC
D3:11122233	001506:EF11
D5:FA000070	001508:2233
D6:AB00FF9B	00150A:4455

Instrucciones:	
CLR.B -(A4)	
MOVE.L -124(A2, D5.W), -(A1)	
MOVE.W $64(A4,D6.W), D3	

13. Comparar los computadores de 1,2 y 3 direcciones escribiendo los programas para calcular la expresión X= (A+B*C)/(D-E*F) siendo los repertorios los siguientes:

```
0 Address       1 Address       2 Address               3 Address
PUSH M          LOAD M          MOVE X,Y ;(X<-Y)        MOVE X,Y ;(X<-Y)
POP M           STORE M         ADD X,Y  ;(X <- X+Y)    ADD X,Y ;(X <- Y+Z)
ADD             ADD M           SUB X,Y  ;(X <- X-Y)    SUB X,Y ;(X <- Y-Z)
SUB             SUB M           MUL X,Y  ;(X <- X*Y)    MUL X,Y ;(X <- Y*Z)
MUL             MUL M           DIV X,Y  ;(X <- X/Y)    DIV X,Y ;(X <- X/Y)
DIV             DIV M
```

- SOLUCION:

```
PUSH A      LOAD   E     MOV R0, E     MUL R0, E, F
PUSH B      MUL    F     MUL RO, F     SUB R0, D, R0
PUSH C      STORE  T     MOV R1, D     MUL R1, B, C
MUL         LOAD   D     SUB R1, R0    ADD R1, A, R1
ADD         SUB    T     MOV R0, B     DIV X, R0, R1
PUSH D      STORE  T     MOV R0, C
PUSH E      LOAD   B     ADD R0, A
PUSH F      MUL    C     DIV R0, R1
MUL         ADD    A     MOV X,  R0
SUB         DIV    T
DIV         STO    X
POP X
```

14. If the last operation performed on a computer with an 8-bit word was an addition in which the two operands were 00000010 and 00000011, what would be the value of the following flags?

- Carry
- Zero
- Overflow
- Sign
- Even Parity
- Half-Carry
- SOLUCION:
  - 0010+0011=0101 → No hay llevada en el MSB, el resultado no es cero, no hay overflow ya que no hay llevada, positivo, número de unos par, no hay llevada en el bit de posición 3. Por lo que todos los flags desactivados excepto el de paridad par . El flag parity estará a 1.

15. The x86 Compare instruction (CMP) subtracts the source operand from the destination operand; it updates the status flags (C, P, A, Z, S, O) but does not alter either of the operands. The CMP instruction can be used to determine if the destination operand is greater than, equal to, or less than the source operand.

    a. Suppose the two operands are treated as unsigned integers. Show which status flags are relevant to determine the relative size of the two integer and what values of the flags correspond to greater than, equal to, or less than.

    b. Suppose the two operands are treated as twos complement signed integers. Show which status flags are relevant to determine the relative size of the two integer and what values of the flags correspond to greater than, equal to, or less than.

    - SOLUCION:

- Ver Programación en Lenguaje Ensamblador (x86),
- CMP=Dest-Source

c. Enteros sin signo

Cuadro 1: CMP/EFLAFS

Caso	CF	PF	AF	ZF	SF	OF
Dest>Source	**0**	**x**	**x**	**0**	**x**	**x**
Dest=Source	**0**	**x**	**x**	**1**	**x**	**x**
Dest<Source	**1**	**x**	**x**	**0**	**x**	**x**

d. Enteros con signo

Cuadro 2: CMP/EFLAFS

Caso	CF	PF	AF	ZF	SF	OF
Dest>Source	**x**	**x**	**x**	**0**	**0**	**0**
Dest=Source	**0**	**x**	**x**	**1**	**0**	**0**
Dest<Source	**1**	**x**	**x**	**0**	**1**	**?**

- ? : habrá o no overflow dependiendo del signo del destino y fuente. Pej operandos de 8 bits con OP_destino=-255 y OP_fuente=+1, entonces OP_destino - OP_fuente = -255-1, en C2 = 0x80+0xFF=0x17F=1_0111_1111= Sí hay oveflow ya que MSB de valor 1 es distinto al bit de signo 0.

16. Many microprocessor instruction SETS include an instruction that tests a condition and sets a destination operand if the condition is true. Examples include the *SETcc* on the x86, the Scc on the Motorola MC68000, and the Scond on the National NS32000.

    a. A simple IF statement such as *IF a > b* THEN can be implemented using a numerical representation method, that is, making the *Boolean value* manifest, as opposed to a *flow of control* method, which represents the value of a Boolean expression by a point reached in the program. (Primero transcribir el lenguaje ASM del enunciado a lenguaje RTL). A compiler might implement *IF a > b THEN* with the following **x86 code**:

```
        ; Sintaxis de Intel: Opeación Op_destino, Op_fuente
        SUB CX, CX ;set register CX to 0
        MOV AX, B ;move contents of location B to register AX
        CMP AX, A ;compare contents of register AX and location A
        JLE TEST ;jump if A <= B
        INC CX ;add 1 to contents of register CX
TEST    JCXZ OUT ;jump if contents of CX equal 0

THEN    XXXXX

OUT     XXXXX
```

    b. Now consider the high-level language statement, (Primero transcribir el lenguaje ASM del enunciado a lenguaje RTL):

    - *A:=(B > C) OR (D == F)*
    - A compiler might generate the following code:

```
        MOV EAX, B ;move contents of location B to register EAX
        CMP EAX, C ;compare contents of register EAX and location C
        MOV BL, 0 ;0 represents false
        JLE N1 ;jump if (B <= C)
        MOV BL, 1 ;1 represents false
N1      MOV EAX, D
        CMP EAX, F
        MOV BH, 0
        JNE N2
```

```
        MOV BH, 1
N2      OR BL, BH
```

- SOLUCION:
- a. `JLE Op_destino`

- Salta si el ultimo resultado activa el banderín ZF=1 ó los banderines SF y OF son diferentes (SF<>OF)
- `CMP Op_destino_B,Op_fuente_A` → B-A → operandos con signo en complemento a dos
  - ZF=1 → B==A
  - SF <> OF
  - SF=1,OF=0 → B<A
  - SF=0,OF=1 → B>A
- Sustituir el salto por SETcc

```
        SUB CX, CX ;set register CX to 0
        MOV AX, B  ;move contents of location B to register AX
        CMP AX, A  ;compare contents of register AX and location A
        SETGT CX   ;CX = (a GT b)
TEST    JCXZ OUT   ;jump if contents of CX equal 0
THEN    XXXXXX
OUT     XXXXXX
```

- b.

```
MOV EAX, B ; move from location B to register EAX
CMP EAX, C
SETG BL ; Setcc SETGreater ;BL = 0/1 depending on result
MOV EAX, D
CMP EAX, F
MOV BH, 0
SETE BH ; Setcc SETEqual ;BH= 0 ó 1 dependiendo del resultado
OR BL, BH
```

17. En la estructura de datos siguiente, dibujar el layout de memoria little-endian, teniendo en cuenta que el compilador alinea los datos con direcciones múltiplo de 4 rellenando los huecos con ceros, y así minimizar el número de transferencias entre la memoria y la CPU en la captura de los datos.

- Declaración:

```
#include <stdio.h>
void main (void)
{
  struct{
    int a;
    int pad; //
    double b;
    int* c;
    char d[7];
    short e;
    int f;
    char q[4];
  } s={.a=0x11121314,.pad=0,.b=0x2122232425262728,.d={'A','B','C','D','E','F','G'},.e ↩
     =0x5152,.f=0x61626364,.q="abc"};
  s.c=&s.e;
}
```

- SOLUCION Little Endian:

```
00: 14 13 12 11
04: xx xx xx xx
08: rr rr rr rr
```

```
0C: rr rr rr rr
10: aa aa aa aa
14: 41 42 43 44
18: 45 46 47 pp
1C: 52 51 pp pp
20: 64 63 62 61
24: 61 62 63 00
```

- xx: indeterminado
- rr: código ieee-754 doble precisión
- aa: dirección de la variable e con la que es inicializado el puntero c.
- char q[4]="abc" : array de 4 elementos tipo carácter. Equivale a: char q[4]={*a*,*b*,*c*,NULL} donde el caracter NULL vale 00.

18. Para una arquitectura little endian el mapa de direcciones en memoria es el de la figura de abajo. Asociar la declaración de las estructuras s1 y s2 en lenguaje C y su inicialización con el mapa de direcciones indicando las direcciones en memoria de los elementos de las estructuras.

./images/ejercicios/little_endian.png

Figura 3: Little Endian

a. declaración de la variable s1: tipo estructura

```
struct {
double i; //0x1112131415161718 ; 8 bytes
} s1;
```

- SOLUCION:
  - MSB(i):0x11 en la dirección 0x03 y LSB(i):18 en la dirección de MSB-7

b. declaración de la variable s2: tipo estructura

```
struct {
int i; //0x11121314 ; 4 bytes
int j; //0x15161718
} s2;
```

- SOLUCION:
- i:
  - MSB(i):0x11 en la dirección 0x03 y LSB(i) en la dirección 0x00
- j:
  - j se reservar secuencialmente a continuación de i, por lo que LSB(j) estará en la dirección 0x04 y MSB(j) en la dirección 0x07
  - 0x04 : 18 17 16 15
    - Write a small program to determine the endianness of machine and report the results. Run the program on a computer available to you and turn in the output.

```
#include <stdio.h>
main()
{
  int integer; /*4 bytes*/
  char *p;
  integer = 0x30313233; /* ASCII for chars '0', '1', '2', '3' */
```

```
p = (char *)&integer
  if (*p=='0' && *(p+1)=='1' && *(p+2)=='2' && *(p+3)=='3')
    printf("This is a big endian machine.\n");
  else if (*p=='3' && *(p+1)=='2' && *(p+2)=='1' && *(p+3)=='0')
    printf("This is a little endian machine.\n");


  else
    printf("Error in logic to determine machine endian-ness.\n");
```

- SOLUCION:
  - p apunta al primer byte de la variable integer, (p+1) al siguiente byte y así sucesivamente
  - si p apunta al caracter *0* significa que el MSB de integer se almacena en la dirección más baja → Big endian
  - si p apunta al caracter *3* significa que el MSB de integer se almacena en la dirección más alta → Little endian

6. Programación asm

- Explicación breve del modus operandi de los códigos mnemónicos. Para información más detallada ir al Manual del Repertoriod e Instrucciones de INTEL

6.1. Datos

1. Interpretar las instrucciones siguientes de un programa en lenguaje ensamblador x86-64 describiéndolas en lenguaje RTL:
   a. mov da1,da4
   b. mov $0xFF00FF00FF00FF00,%rax
   c. mov $0xFF,%rsi
   d. mov $da1,%rsp
   e. lea da1,%rsp
   f. mov da4,%ebx
   g. movb da4,%ebx
   h. movl da4,%ax
   i. movw %ebx,da4
   j. movw %ebx,da1
      - si la sección de datos presenta el código siguiente:

```
        .data

da1:    .byte   0x0A
da2:    .2byte  0x0A0B
da4:    .4byte  0x0A0B0C0D
saludo: .ascii  "hola"
lista:  .int    1,2,3,4,5
```

      - SOLUCION

```
mov da1,da4
mov $0xFF00FF00FF00FF00,%rax
mov $0xFF,%rsi
mov $da1,%rsp
lea da1,%rsp
mov da4,%ebx
movb da4,%ebx
movl da4,%ax
movw %ebx,da4
movq %ebx,da4
```

6.2. Modos de Direccionamiento

1. Deducir la dirección efectiva del operando en las expresiones siguientes:
   a. $0
   b. %rax
   c. loop_exit
   d. data_items(,%rdi,4)
   e. (%rbx)
   f. (%rbx,%rdi,4)

- SOLUCIONES
  - $0 : inmediato : el operando está en la propia instrucción, es 0.
  - %rax : directo registro . El operando está en el registro. Operando R[rax]
  - loop_exit : directo memoria . La etiqueta es la dirección efectiva del operando en memoria. Operando M[loop_exit]
  - data_items(,%rdi,4) : indexado y desplazamiento inmediato. Dirección efectiva = data_item+4*RDI. Operando M[data_item+4*RDI]
  - (%rbx) : Indirecto a registro . Dirección efectiva=RBX . Operando M[RBX]
  - (%rbx,%rdi,4) : indexado y desplazamiento en registro base. Dirección efectiva = RBX+4*RDI .Operando M[RBX+4*RDI]

2. Describir en lenguaje RTL el código

```
lea buffer,%eax
mov da2,(%eax)
mov da2,%bx
mov %bx, (%eax)
incw da2
lea  da2,%ebx
incw 2(%ebx)
mov $3,%esi
mov da2(,%esi,2),%ebx
```

- SOLUCION:

```
lea buffer,%eax
mov da2,(%eax)
mov da2,%bx
mov %bx, (%eax)
incw da2
lea  da2,%ebx
incw 2(%ebx)
inc 2(%ebx)
mov $3,%esi
mov da2(,%esi,2),%ebx
```

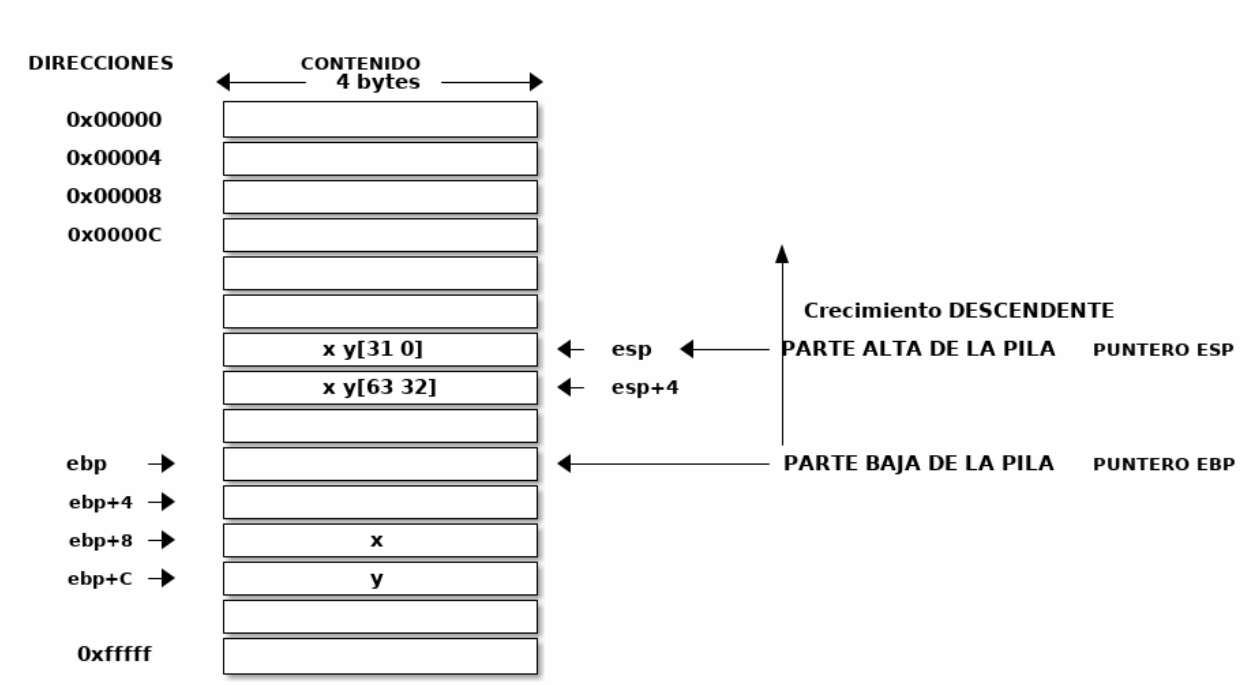
6.3. Aritmética

1. Suponer que los números enteros con signo x e y están almacenados en las posiciones 8 y 12 relativas al registro %ebp , y se desea almacenar en el top de la pila el producto x*y de 8 bytes, siendo el registro %esp (stack pointer) el puntero al top de la pila. a)Desarrollar el código ensamblador gas para la arquitectura i386 y b) dibujar el contenido de la pila sabiendo que la anchura de la pila es una palabra en la arquitectura i386 y suponiendo que el registro *ebp* y el puntero de pila *esp* están distanciados 4 palabras.

- SOLUCION: Módulo asm:

```
x en %ebp+8, y en %ebp+12

1 movl 12(%ebp), %eax
2 imull 8(%ebp)                          ; EDX:EAX<-x*y[63-32:31-0] ; imul:  ↩
    multiplicación de enteros con signo
3 movl %eax, (%esp)                      ; pila <-x*y[31:0]
4 movl %edx, 4(%esp)                     ; pila <-x*y[63:32]
```

- Pila: arquitectura i386 → x86-32 → palabra=4bytes. EL crecimiento de la pila es hacia direcciones de la memoria más bajas.



- EBP: base pointer register: apunta al bottom de la pila
- ESP: stack pointer register: apunta al top de la pila
- La pila está ocupada desde la dirección bottom hasta la dirección top, donde la dirección top < dirección bottom.

2. Suponer que los números enteros con signo x e y están almacenados en las posiciones 8 y 12 relativas al registro %ebp , y se desea almacenear en el top de la pila el producto x/y y también x mod y, siendo el registro %esp (stack pointer) el puntero al top de la pila. Desarrollar el código ensamblador gas para la arquitectura i386.

   - Solución: Módulo asm

```
x en ebp+8, y en ebp+12

1 movl 8(%ebp), %edx    ; edx<-x
2 movl %edx, %eax       ; copiar x en eax
3 sarl $31, %edx        ; edx lo lleno con el bit de signo de x , ya que forma parte  ↩
    del dividendo.
                        ; edx:eax <- x
4 idivl 12(%ebp)        ; EAX<-Cociente{x/y} , EDX<-Resto{x/y]}
5 movl %eax, 4(%esp)    ; pila <- Cociente{x/y}
6 movl %edx, (%esp)     ; pila <- x%y  ; x%y = Resto{x/y}
```

   - Mismo enunciado anterior pero utilizando la instrucción de extensión de signo `cltd`
     - Módulo asm

```
x en ebp+8, y en ebp+12

1 movl 8(%ebp), %edx    ; edx<-x
2 movl %edx, %eax       ; copiar x en eax
3 cltd                  ; extiende el signo del operando en eax a edx
4 idivl 12(%ebp)        ; EAX<-Cociente{x/y} , EDX<-Resto{x/y]}
5 movl %eax, 4(%esp)    ; pila <- Cociente{x/y}
6 movl %edx, (%esp)     ; pila <- x%y  ; x%y = Resto{x/y}
```

3. Del módulo fuente en lenguaje C:

- Deducir el tipo num_t del argumento del prototipo de la función store_prod
- Interpretar el módulo ASM en lenguaje RTL

```
void store_prod(num_t *dest, unsigned x, num_t y) {
*dest = x*y;
}
```

```
dest en ebp+8, x en ebp+12, y en ebp+16

1 movl 12(%ebp), %eax
2 movl 20(%ebp), %ecx
3 imull %eax, %ecx
4 mull 16(%ebp)
5 leal (%ecx,%edx), %edx
6 movl 8(%ebp), %ecx
7 movl %eax, (%ecx)
8 movl %edx, 4(%ecx)
```

- SOLUCION:
  - El argumento dest está implementado en la dirección ebp+8
  - dest es un puntero a un objeto de tipo num_t
  - línea 6: carga ecx con el valor de dest
  - línea 7: carga eax en la dirección de memoria a la que apunta dest
  - línea 1: carga eax con la variable x que es de tipo sin signo, luego num_t es unsigned.

6.4. Saltos

- Calcular las direcciones de salto en código máquina en el siguiente bloque de código ensamblador:
  - módulo fuente:

```
1 jle .L2 if <=, goto dest2
2 .L5: dest1:
3 movl %edx, %eax
4 sarl %eax
5 subl %eax, %edx
6 leal (%edx,%edx,2), %edx
7 testl %edx, %edx
8 jg .L5 if >, goto dest1
9 .L2: dest2:
10 movl %edx, %eax
```

  - Módulo objeto reubicable :las posiciones de memoria son relativas a la dirección de referencia "silly" (dirección cero del módulo reubicable)

```
1       8: 7e 0d jle 17 <silly+0x17> Target = dest2
2       a: 89 d0 mov %edx,%eax dest1:
3       c: d1 f8 sar %eax
4       e: 29 c2 sub %eax,%edx
5       10: 8d 14 52 lea (%edx,%edx,2),%edx
6       13: 85 d2 test %edx,%edx
7       15: 7f f3 jg a <silly+0xa> Target = dest1
8       17: 89 d0 mov %edx,%eax dest2:
```

  - Módulo objeto ejecutable : El linker ha resuelto las posiciones de memoria relativas del módulo objeto reubicable convirtiendolas en direcciones de memoria absolutas.

```
1 804839c: 7e 0d jle 80483ab <silly+0x17>
2 804839e: 89 d0 mov %edx,%eax
3 80483a0: d1 f8 sar %eax
4 80483a2: 29 c2 sub %eax,%edx
5 80483a4: 8d 14 52 lea (%edx,%edx,2),%edx
6 80483a7: 85 d2 test %edx,%edx
7 80483a9: 7f f3 jg 804839e <silly+0xa>
8 80483ab: 89 d0 mov %edx,%eax
```

- SOLUCION:
- los saltos están en las líneas 1 y 7 del código.
- el operando del salto de la línea 1 es la dirección absoluta 80483ab etiquetada como dest2
  - Cuando se ejecuta la línea 1 el PC ó RIP apunta a la línea 2, es decir, 804839e
  - El salto será la resta 80483ab - 804839e = 0D
- el operando del salto de la línea 7 es la dirección 804839e etiquetada como dest1
  - El salto será la resta 804839e - 80483ab
    - Como la resta va a dar negativo invierto los operandos y después cambio el signo del resultado
    - 80483ab - 804839e = 0D
    - El complemento a 2 de tamaño 1 byte de 0D es F2+1 = F3

6.5. If-Then-Else

1. Relacionar un programa en lenguaje C (Ref. Randal194) y el programa equivalente en lenguaje ensamblador mediante comentarios en el módulo fuente de bajo nivel:

   - Módulo C:

```
(a) Original C code
1 int absdiff(int x, int y) {
2 if (x < y)
3 return y - x;
4 else
5 return x - y;
6 }

(b) Equivalent goto version
1 int gotodiff(int x, int y) {
2 int result;
3 if (x >= y)
4 goto x_ge_y;
5 result = y - x;
6 goto done;
7 x_ge_y:
8 result = x - y;
9 done:
10 return result;
11 }
```

   - Módulo ASM:

```
(c) Generated assembly code
x at %ebp+8, y at %ebp+12
1 movl 8(%ebp), %edx Get x
2 movl 12(%ebp), %eax Get y
3 cmpl %eax, %edx Compare x:y
4 jge .L2 if >= goto x_ge_y
5 subl %edx, %eax Compute result = y-x
```

```
6 jmp .L3 Goto done
7 .L2: x_ge_y:
8 subl %eax, %edx Compute result = x-y
9 movl %edx, %eax Set result as return value
10 .L3: done: Begin completion code
```

6.6. Do-While Loops

1. Relacionar un programa en lenguaje C (Ref. Randal199) y el programa equivalente en lenguaje ensamblador mediante comentarios en el módulo fuente de bajo nivel:

   - Módulo C:

```
1 int dw_loop(int x, int y, int n) {
2 do {
3 x += n;
4 y *= n;
5 n--;
6 } while ((n > 0) && (y < n));
7 return x;
8 }
```

   - Módulo ASM:

```
x at %ebp+8, y at %ebp+12, n at %ebp+16

1 movl 8(%ebp), %eax
2 movl 12(%ebp), %ecx
3 movl 16(%ebp), %edx
4 .L2:
5 addl %edx, %eax
6 imull %edx, %ecx
7 subl $1, %edx
8 testl %edx, %edx
9 jle .L5
10 cmpl %edx, %ecx
11 jl .L2
12 .L5:
```

     - Write a goto version of the function (in C) that mimics how the assembly code program operates.

2. Relacionar un programa en lenguaje C (Ref. Randal201) y el programa equivalente en lenguaje ensamblador mediante comentarios en el módulo fuente de bajo nivel:

   - Módulo C:

```
1 int loop_while(int a, int b)
2 {
3 int result = 1;
4 while (a < b) {
5 result *= (a+b);
6 a++;
7 }
8 return result;
9 }
```

   - Ensamblaje: In generating the assembly code, gcc makes an interesting transformation that, in effect, introduces a new program variable. Register %edx is initialized on line 6 and updated within the loop on line 11. Describe how it relates to the variables in the C code. Create a table of register usage for this function.

```
a at %ebp+8, b at %ebp+12

1 movl 8(%ebp), %ecx
2 movl 12(%ebp), %ebx
3 movl $1, %eax
4 cmpl %ebx, %ecx
5 jge .L11
6 leal (%ebx,%ecx), %edx
7 movl $1, %eax
8 .L12:
9 imull %edx, %eax
10 addl $1, %ecx
11 addl $1, %edx
12 cmpl %ecx, %ebx
13 jg .L12
14 .L11:
```

3. Relacionar un programa en lenguaje C (Ref. Randal204) y el programa equivalente en lenguaje ensamblador mediante comentarios en el módulo fuente de bajo nivel:

   - Módulo C:

```
1 int fact_for_goto(int n)
2 {
3 int i = 2;
4 int result = 1;
5 if (!(i <= n))
6 goto done;
7 loop:
8 result *= i;
9 i++;
10 if (i <= n)
11 goto loop;
12 done:
13 return result;
14 }
```

   - Módulo asm

```
Argument: n at %ebp+8
Registers: n in %ecx, i in %edx, result in %eax

1 movl 8(%ebp), %ecx Get n
2 movl $2, %edx Set i to 2 (init)
3 movl $1, %eax Set result to 1
4 cmpl $1, %ecx Compare n:1 (!test)
5 jle .L14 If <=, goto done
6 .L17: loop:
7 imull %edx, %eax Compute result *= i (body)
8 addl $1, %edx Increment i (update)
9 cmpl %edx, %ecx Compare n:i (test)
10 jge .L17 If >=, goto loop
11 .L14: done:
```

7. Lenguaje de Programación C

7.1. Punteros

1. Editar y ejecutar el siguiente programa en lenguaje C interpretando el resultado sabiendo que lista es una "variable puntero" que apunta al elemento lista[0].

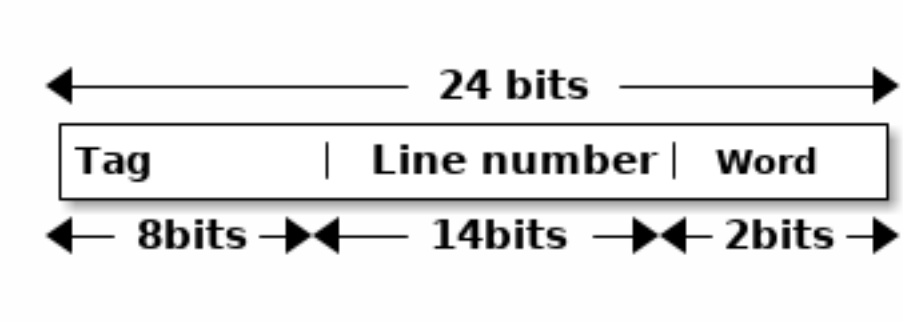
```
#include <stdio.h>

void main (void)
{

  int lista[]={1,2,3,4,5};

  printf ("\n **********ARRAY LISTA***********");
  printf ("\n *******lista es una VARIABLE PUNTERO*******");
  printf ("\n **la variable lista contiene la dirección de lista[0]*******\n\n") ;
  printf("\n lista[0] = %d es el 1° elemento de lista \n", lista[0]);
  printf("\n lista = %p es la dirección del 1° elemento \n", lista);
  printf("\n &lista[0] = %p es la dirección del 1° elemento \n", &lista[0]);
  printf("\n *lista = %d equivale a lista[0] \n", *lista);
  printf("\n *&lista = %p equivale a lista \n", *&lista);
  printf("\n **&lista = %d equivale a lista[0] \n", **&lista);

  printf ("\n\n *******ARITMETICA DE PUNTEROS*******");
  printf("\n *(lista+1) = %d equivale a lista[1] \n", *(lista+1));
  printf("\n *(lista+4) = %d equivale a lista[4] \n", *(lista+4));

  printf ("\n\n *******CASTING*******");
  printf("\n (int *)lista = %p  \n", (int *)lista);
  printf("\n *(int *)lista = %d \n", *(int *)lista);
  printf("\n *(int)lista+1 = %d \n", *(int *)lista+1);
  printf("\n *(int)lista+4 = %d \n", *(int *)lista+4);

  printf("\n *((short *)lista+1) = %d . Ahora lista lo declaro con elementos de 2   ↩
      bytes \n", *((short *)lista+1));
}
```

2. Al depurar con el depurador GDB un programa escrito en lenguaje ensamblador donde se ha declarado la directiva `lista: .int 1,2,3,4,5`, deducir las siguientes sentencias si la etiqueta lista es ensamblada como la dirección 0x55bf0000 y por lo tanto no es una variable puntero, sino que es el mismo puntero, es decir, la dirección del primer elemento del array lista. Indicar la relación entre el mapa de direcciones de memoria y el mapa de posiciones de elementos del array lista.

```
&lista  : dirección de la "variable array" lista para el elemento de la posición 0  : 0 ↩
    x55bf0000
&lista+1                : 0x55bf0004 -> dirección para el elemento de la posición 1.
(void *)&lista+1        : 0x55bf0001
(int  *)&lista+1        : 0x55bf0004
lista                   : 1 -> equivale a lista[0]
(int)lista              : 1
(int *)&lista           : 0x55bf0000
(int [5])lista          : {1,2,3,4,5}
*(int *)&lista+1        : 2
```

8. Capitulo 4: Memoria Cache

- Ejemplo 4.2a Pg118. The system has a Cache memory of 64KB and Main Memory of 16MB with a byte word size and four word block size. For a cache controller with direct mapping correspondence function search the main memory block addresses correspondences to cache memory 0x0CE7 number line .
  - Desarrollo:
    - Memoria principal: 16MB, byte word, 4 byte block.
      - ◇ 16MB → 2^{24} → 24 bits address bus
    - Memoria cache: 64KB, 4 byte line, 16K lines.
      - ◇ 16K → 2^{14}→ 14 bits campo de línea
    - Direct mapping correspondence function: 0x0CE7 cache line
    - i=j mod m donde i es el número de línea, j el número de bloque y m el número de líneas de la caché.
    - la dirección de 24 bits se descompone en : etiqueta-línea-palabra



    - Cada tag agrupa 16K bloques → 16K bloquesx4bytes/bloquexnúmero de tags N=16MB → 2^{14}x2^{2}xN=2^{24}bytes → N=2^{8} Tags
    - 0CE7 : 14 bits: 00-1100-1110-0111. Buscamos las direcciones de memoria asociadas a dicha línea.
      - ◇ Tag 0, Linea 0CE7, Palabra 0 → 0000-0000-00-1100-1110-0111-00 = 0000-0000-0011-0011-1001-1100 → 00339C
      - ◇ Tag 1, Línea 0CE7, Palabra 0 → cambia el primer dígito a 1 → 01339C
      - ◇ Tag 255, Línea 0CE7, Palabra 0 → cambia el primer dígito a FF → FF339C
      - ◇ Las direcciones de memoria son la dirección de la primera palabra de bloque en Memoria Principal.

- 4.1 A set-associative cache consists of 64 lines, or slots, divided into four-line sets. Main memory contains 4K blocks of 128 words each. Show the format of main memory addresses.
  - Desarrollo:
    - Caché: 64 líneas de 128 palabras cada una agrupadas en sets de 4 líneas
      - ◇ 128 palabras → 7 bits para direccionar la palabra dentro de la línea
      - ◇ 16 sets → 2^{4} → 4 bits para direccionar los sets dentro de la caché
    - Main memory: 4Kblocks de 2^{7}palabras
      - ◇ 12 bits para direccionar un bloque
      - ◇ 2^{19} palabras→512Kpalabras → 19 bits para direccionar una palabras → ancho bus de direcciones
    - set associative → i = j mod v donde v es el número de sets, j el bloque e i el set
      - ◇ Tag → código para diferenciar los bloques que van al mismo set. bits Tag=bits totales - bits Set - bits Word=19-4-7=8 bits.
    - Tag/Set/Word → 19 address bits descompuestos en los 3 campos de 8/4/7 bits
  - Sol:
    - Tag/Set/Word : 8/4/7

- 4.3 For the hexadecimal main memory addresses 111111, 666666,BBBBBB, show the following information, in hexadecimal format:

a. Tag, Line, and Word values for a direct-mapped cache, using the format of Figure 4.10
b. Tag and Word values for an associative cache, using the format of Figure 4.12
c. Tag, Set, and Word values for a two-way set-associative cache, using the format of Figure 4.15

- Desarrollo:
  - a) Direct mapped Tag/Line/Word → 24 address bits descompuestos en los 3 campos de 8/14/2 bits
    - ◇ 111111 = 0001-0001-0001-0001-0001-0001 = 00010001-00010001000100-01=0001-0001-00-0100-0100-0100-01=11-0444-1 → El 0 no se escribe en hex por la izda
  - b) Full associative cache Tag/Word → 24 address bits descompuestos en los 2 campos de 22/2 bits
    - ◇ 111111 = 0001-0001-0001-0001-0001-0001 = 000100010001000100100-01=00-0100-0100-0100-0100-0100-01=044444-1 → El 0 no se escribe en hex por la izda
  - c) Set associative cache Tag/Set/Word → 24 address bits descompuestos en los 3 campos de 9/13/2
    - ◇ 111111 = 0001-0001-0001-0001-0001-0001 = 000100010-0010001000100-01=0-0010-0010-0-0100-0100-0100-01=022/0444/1-1 → El 0 no se escribe en hex por la izda
- Sol:

Cuadro 3: Direcciones

Address	**111111**	**666666**	**BBBBBB**
a. Tag/Line/Word	11/444/1	66/1999/2	BB/2EEE/3
b. Tag/Word	44444/1	199999/2	2EEEEE/3
c. Tag/Set/Word	22/444/1	CC/1999/2	177/EEE/3

- 4.5 Consider a 32-bit microprocessor that has an on-chip 16-KByte four-way set-associative cache. Assume that the cache has a line size of four 32-bit words.

a. Draw a block diagram of this cache showing its organization and how the different address fields are used to determine a cache hit/miss.
b. Where in the cache is the word from memory location ABCDE8F8 mapped

- Desarrollo:
  - Memoria principal
    - ◇ no dice nada del bus externo, supongo el máximo de 32 bits → 2^{32}Bytes → 4GB
  - Cache on-chip: bus local: 32 bits data bus y address bus: Set associative de 4 líneas por set.
    - ◇ 4 palabras de 4 bytes cada una por línea hacen un total de 16 bytes por línea (4 bits en el campo word). El código de 4 bits direcciona el primer byte de cada palabra (0x0 la palabra 0, 0x4 la palabra 1, 0x8 la palabra 2, 0xC la palabra 3)
    - ◇ El número de sets es capacidad total/bytes por set = 16KB / (4líneas/set)**(4palabras/línea)**(4bytes/palabra) = 16KB/64B = 2^8=256 sets → 8bits
    - ◇ el número de bloques en cache es capacidad/bytes_por_línea= 16KB/(4palabras/línea)*(4bytes/palabra)=1Kbloques
    - ◇ los 1kbloques se asocian en sets de 4 líneas.
    - ◇ address bus=tag bits+set bits+word bits → 32=tag bits+8+4 → tag_bits=32-8-4=20 bits. El campo Tag distingue bloques dentro del mismo set.
    - ◇ ¿que bloques van al mismo set? i = j mod v, donde i es el número de set al que va el bloque j, v es el número de sets. Es decir, 2^20 bloques están asocidos al mismo set por lo que han de compartir 4 líneas → 4 para 20.
    - ◇ Tag/Set/Word → 32 address bits descompuestos en los 3 campos de 20/8/4 bits
  - a) Después de la descomposición tag/set/word se selecciona el set direccionado y se comparan los tags de las 4 líneas con el tag de la dirección absoluta. Son 4 comparadores, uno por vía. Al Comparador_1 irán la primera línea de cada set en que dividimos la memoria principal. 2^{32} bytes los agrupo en sets de 16 palabras por set → la memoria principal queda dividida en 2^8 sets

    - b) Descomposición 20/8/4 de la dirección ABCD8F8→ ABCD/8F/8 → 8F es el set 143 y el byte 8 es la palabra número 2.
  - Sol: a… Descomposición: Tag/Set/Offset . 4 comparadores: 1 por cada vía del Set. b… Set 143, cualquier línea, la doblepalabra número 2.

- 4.7 The Intel 80486 has an on-chip, unified cache. It contains 8 KBytes and has a four-way set-associative organization and a block length of four 32-bit words. The cache is organized into 128 sets. There is a single "line valid bit" and three bits, B0, B1, and B2 (the "LRU" bits), per line. On a cache miss, the 80486 reads a 16-byte line from main memory in a bus memory read burst.

  a. Draw a simplified diagram of the cache
  b. show how the different fields of the address are interpreted.
     - Desarrollo:
       - Intel 80486 (Pag 38,47,130) tiene un bus de memoria de 32 bits → address bus de 32 bits que direccionan 1 byte.
       - Caché: 8KB, set-associative de 4 vías, cada línea 4 palabras de 4 bytes (16 bytes con 4 bits), y 128 sets (7 bits)
         - 4 palabras de 4 bytes cada una por línea hacen un total de 16 bytes por línea (4 bits en el campo word). El código de 4 bits direcciona el primer byte de cada palabra (0x0 la palabra 0, 0x4 la palabra 1, 0x8 la palabra 2, 0xC la palabra 3)
       - Descomposición de los 32 bits : Tag/Set/Offset → 21/7/4
       - Además de los 32 bits es necesario añadir:
         - 3 bits USO para indicar de las cuatro líneas quien es la MENOS recientemente utilizada, la de menor valor de los 8 posibles: 000-001-010-011-100-101-110-111
         - 1 bit de validación que indica con el valor 1 que hace falta su actualización en MP antes de sobreescribir la línea → técnica de postescritura.
       - Línea: Valid/LRU/Tag/Set/Offset → 1/3/21/7/4
     - Solución: a.. Esquema Set associative b.. Valid/LRU/Tag/Set/Offset → 1/3/21/7/4

- 4.15 Consider the following code:

```
for (i=0; i=20; i++)
   for ( j=0; j=10; j++)
         a[i] = a[i] * j ;
```

  a. Give one example of the spatial locality in the code.
  b. Give one example of the temporal locality in the code.
     - Desarrollo
       - en el bucle interno siempre se repite la misma instrucción, siempre accedes a la misma dirección donde esta la instrucción → localidad espacial
       - en el bucle interno siempre se repite la misma instrucción, el futuro es el presente → localidad temporal
       - en el bucle interno con j=0 accedes al operando a[0] y en la siguiente iteracción se repite el mismo operando a[0] → localidad espacial y temporal.

- 4.18 Consider a cache of 4 lines of 16 bytes each. Main memory is divided into blocks of 16 bytes each. That is, block 0 has bytes with addresses 0 through 15, and so on. Now consider a program that accesses memory in the following sequence of addresses:

```
Once: 63 through 70
Loop ten times: 15 through 32; 80 through 95
```

  a. Suppose the cache is organized as direct mapped. Memory blocks 0, 4, and so on are assigned to line 1; blocks 1, 5, and so on to line 2; and so on. Compute the hit ratio.
  b. Suppose the cache is organized as two-way set associative, with two sets of two lines each. Even-numbered blocks are assigned to set 0 and odd-numbered blocks are assigned to set 1. Compute the hit ratio for the two-way set-associative cache using the least recently used replacement scheme.

- 4.21 Consider a single-level cache with an access time of 2.5 ns, a line size of 64 bytes, and a hit ratio of H = 0.95. Main memory uses a block transfer capability that has a firstword (4 bytes) access time of 50 ns and an access time of 5 ns for each word thereafter.

    a. What is the access time when there is a cache miss? Assume that the cache waits until the line has been fetched from main memory and then re-executes for a hit.

    b. Suppose that increasing the line size to 128 bytes increases the H to 0.97. Does this reduce the average memory access time?

- 4.24 On the Motorola 68020 microprocessor, a cache access takes two clock cycles. Data access from main memory over the bus to the processor takes three clock cycles in the case of no wait state insertion; the data are delivered to the processor in parallel with delivery to the cache.

    a. Calculate the effective length of a memory cycle given a hit ratio of 0.9 and a clocking rate of 16.67 MHz.

    b. Repeat the calculations assuming insertion of two wait states of one cycle each per memory cycle.What conclusion can you draw from the results?

- 4.27 For a system with two levels of cache, define Tc1 first-level cache access time; Tc2 second-level cache access time; Tm memory access time;H1 first-level cache hit ratio;H2 combined first/second level cache hit ratio. Provide an equation for Ta for a read operation.

9. Capitulo 5: Memoria Sincrona Dinamica RAM (SDRAM)

- 5.x La arquitectura de un computador Intel tiene un bus del sistema con una frecuencia de reloj de 100MHz, el ancho del bus de datos son 64 bits y el ancho del bus de direcciones de la placa base es de 48 bits.

  a. Calcular el ancho de banda del bus en transferencias/s y en bytes/s

  b. Calcular la capacidad de memoria

  c. Calcular el ciclo de memoria teniendo en cuenta que la latencia de la memoria DRAM son 10ns.

  - Desarrollo
    I. 100 x 10^6 ciclos/seg x 1Transferencia/ciclo = 100MT/s
    II. bus de direcciones $\rightarrow$ 2^{48} Words = 2^8 x 2^{40} = 256 TWords = 2^{48} x 2^3 Bytes = 2 x 2^{50} = 2 petabytes
    III. ciclo de memoria ideal (sin bus multiplexado, sin precarga, etc)= 1 Transferencia= latencia_memoria + latencia_bus_transferencia = 10ns + 1/(10^8) = 10ns + 10ns = 20 ns

- 5.2 Consider a dynamic RAM that must be given a refresh cycle 64 times per ms. Each refresh operation requires 150 ns; a memory cycle requires 250 ns.What percentage of the memory's total operating time must be given to refreshes?

  - Desarrollo
    - en 1 ms 64 refrescos de 150ns $\rightarrow$ 9600 ns refrescando
    - 9600ns/1ms = 0.0096 = 1 %
  - Sol:
    a. 1 %

- 5.3 Figure 5.16 shows a simplified timing diagram for a DRAM read operation over a bus. The access time is considered to last from t1 to t2. Then there is a recharge time, lasting from t2 to t3, during which the DRAM chips will have to recharge before the processor can access them again.

  a. Assume that the access time is 60 ns and the recharge time is 40 ns.What is the memory cycle time? What is the maximum data rate this DRAM can sustain, assuming a 1-bit output?

  b. Constructing a 32-bit wide memory system using these chips yields what data transfer rate?

  - Desarrollo:
  - t1$\rightarrow$t2 : direccionamiento
  - t2$\rightarrow$t3
    - acceso al dato
    - recarga del bus de direcciones a medio camino entre el 0 y el 1
    - 60ns de latencia y 40 de precarga = 100 ns de ciclo de memoria entre 2 lecturas consecutivas
      - durante la precarga se realizaría el burst que puede ser mayor, menor o igual a la precarga
    - El ciclo de bus de 100 ns son 1/100ns= 10MHz. Si transferimos un bit por ciclo de bus= 10Mbps
    - Si utilizamos 32 líneas en paralelo = 32 bits/transferencia x 10MT/s = 320Mbps = 40 MB/s
  - Sol:
    I. tcycle = 100ns. BW=10Mbps
    II. 40MB/s

- 5.4 Figure 5.6 indicates how to construct a module of chips that can store 1 MByte based on a group of four 256-Kbyte chips. Let's say this module of chips is packaged as a single 1-Mbyte chip, where the word size is 1 byte. Give a high-level chip diagram of how to construct an 8-Mbyte computer memory using eight 1-Mbyte chips. Be sure to show the address lines in your diagram and what the address lines are used for.

  - Desarrollo:
    - Con 4 chips de 256Kbit creo un módulo-chip de 1Mb

- Con 8 chips de 1Mb creo un módulo de 8Mb llevando distintos chip select a cada chip de 1Mb. Para seleccionar 1 chip de 8 necesito 3 bits de direcciones, por ejemplo los 3 bits de mayor posición. Para direccionar un bit de un chip 1Mb necesito un bus de direcciones de 20 bits. En total necesito un bus de 20+3=23 bits.

- Sol:

I. 8 chips x1 de capacidad 1M donde cada entrada chip-select es la salida de un decodificador de 3 líneas de dirección

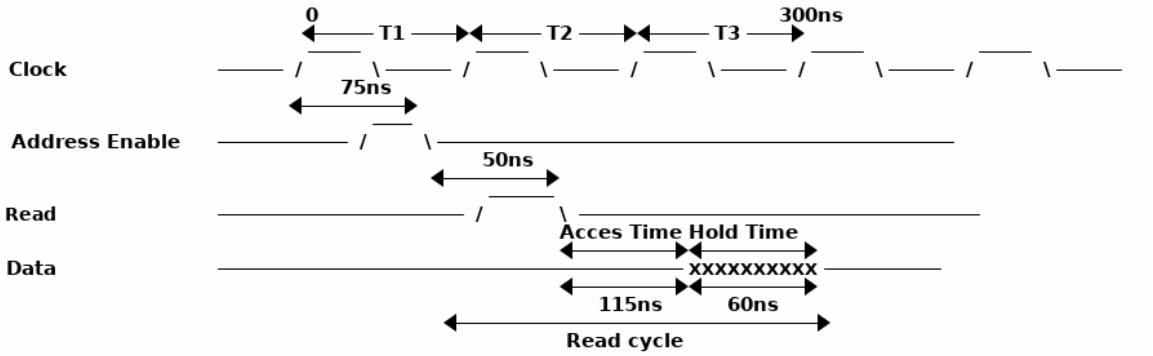
- 5.5 On a typical Intel 8086-based system, connected via system bus to DRAM memory, for a read operation, RAS is activated by the trailing edge of the Address Enable signal (Figure 3.19). However, due to propagation and other delays, RAS does not go active until 50 ns after Address Enable returns to a low. Assume the latter occurs in the middle of the second half of state T1 (somewhat earlier than in Figure 3.19). Data are read by the processor at the end of T3. For timely presentation to the processor, however, data must be provided 60 ns earlier by memory. This interval accounts for propagation delays along the data paths (from memory to processor) and processor data hold time requirements. Assume a clocking rate of 10 MHz.

I. How fast (access time) should the DRAMs be if no wait states are to be inserted?

II. How many wait states do we have to insert per memory read operation if the access time of the DRAMs is 150 ns?

- Desarrollo:
  - Ciclo de lectura
    - Load address - Address Enable (EA)- Address Command - Access Data
    - Trailing edge = fall edge = negative edge
    - AE fall = en la segunda mitad del ciclo T1. Instante 75ns
    - RAS = Read Command : Retardo de 50ns respecto de AE fall. Instante 75+50=125ns
    - La presentación del dato en el bus debe ser realizada con 60 ns de antelación a la carga del dato en la CPU la final del ciclo T3(300ns), es decir, 300ns-60ns=240ns
    - Reloj del bus del sistema = 10 MHz = 100 ns.

A. Tiempo de acceso (desde la orden de lectura hasta volcar el dato la memoria) sin estados de espera = Tiempo de acceso mínimo impuesto por los retardos de la ruta de datos (CPU y bus): 240ns - 125ns = 115ns



B. Si la memoria DRAM tiene un tiempo de acceso 150ns, superior a un ciclo de bus, desde la orden de lectura la cpu debe de esperar dos ciclos de reloj, uno el propio ciclo de la orden de lectura y otro ciclo extra o ciclo de ESPERA. Por lo que 200ns son suficientes para superar los 150ns del tiempo de acceso. Si el ciclo de espera comienza después de los 115ns, tenemos 215ns que superan a los 150ns.

- Sol:

III. 115ns

IV. 1

10. Capitulo 7: Sistemas Entrada/Salida

- 7.1 On a typical microprocessor, a distinct I/O address is used to refer to the I/O data registers and a distinct address for the control and status registers in an I/O controller for a given device. Such registers are referred to as ports. In the Intel 8088, two I/O instruction formats are used. In one format, the 8-bit opcode specifies an I/O operation; this is followed by an 8-bit port address. Other I/O opcodes imply that the port address is in the 16-bit DX register. How many ports can the 8088 address in each I/O addressing mode? .
  - Desarrollo:
    - memory mapped i/o : se reservan direcciones RAM para i/o
    - controlador i/o: registros datos, estado y control : puerto
    - 2 formatos
      - CodOP/Address(Dir Directo) : 8bits/8bits
      - CodOP/DX Register(Dir Indirecto) : 8bits/8bits $\rightarrow$ DX:16bits
      - Número de puertos : Directo $\rightarrow 2^8$ e Indirecto $\rightarrow 2^{16} \Rightarrow$ total= 256+65536=65792 ports
  - Sol:
    - 65792 puertos
- 7.2 A similar instruction format is used in the Zilog Z8000 microprocessor family. In this case, there is a direct port addressing capability, in which a 16-bit port address is part of the instruction, and an indirect port addressing capability, in which the instruction references one of the 16-bit general purpose registers, which contains the port address. How many ports can the Z8000 address in each I/O addressing mode?
  - Desarrollo
    - Modo directo: 2^{16} = 64K = 65536 ports
    - Modo indirecto: 2^{16} = 64 K = 65536 ports
  - Sol
    - 128K=131072 puertos
- 7.5 A system is based on an 8-bit microprocessor and has two I/O devices. The I/O controllers for this system use separate control and status registers. Both devices handle data on a 1-byte-at-a-time basis.The first device has two status lines and three control lines.The second device has three status lines and four control lines.
  a. How many 8-bit I/O control module registers do we need for status reading and control of each device?
  b. What is the total number of needed control module registers given that the first device is an output-only device?
  c. How many distinct addresses are needed to control the two devices?
     - modelo: El controlador i/o de los perifericos tiene implementados los puertos que son la interfaz con el periférico. Los puertos son direccionables y estan formados por un banco de registros: registro de datos, registro de control, registro de estado.
- Desarrollo:
  - buffer data de 8 bits: 2 puertos de datos (in,out) por cada periférico.
  - 1 buffer status de lectura (registro de estado) y 1 buffer control de escritura (registro de control) por cada periférico
  - las líneas de estado y control no son líneas de direccionamiento,sino que serán líneas conectadas a sus respectivos puertos. Las líneas de estado a 1 registro de estado y las líneas de control a 1 registro de control.
    a. : 1 registro de control y 1 registro de estado por cada periférico
    b. : periférico A (1Data+1Status+1Control) y periférico B (2Data+1Status+1Control) = 7 registros
    c. : tantas direcciones como registros = 7 direcciones
- Sol:

a. 1 reg control y 1 reg estado
b. 7 registros
c. 7 direcciones

- 7.6 For programmed I/O, Figure 7.5 indicates that the processor is stuck in a wait loop doing status checking of an I/O device. To increase efficiency, the I/O software could be written so that the processor periodically checks the status of the device. If the device is not ready, the processor can jump to other tasks. After some timed interval, the processor comes back to check status again.

  ./images/ejercicios/7-5.png

  a. Consider the above scheme for outputting data one character at a time to a printer that operates at 10 characters per second (cps).What will happen if its status is scanned every 200 ms?
  b. Next consider a keyboard with a single character buffer. On average, characters are entered at a rate of 10 cps. However, the time interval between two consecutive key depressions can be as short as 60 ms. At what frequency should the keyboard be scanned by the I/O program?

- Desarrollo:

  a. 10cps → El periférico necesita transmitir 1 caracter cada 100ms y escribe en el puerto dicho dato. Si la CPU no salva el dato escrito por el periférico antes de cada escritura, los datos se pierden. Si la CPU consulta cada 200ms y el periférico escribe cada 100ms, cada dos datos uno se pierde. La solución sería aumentar el buffer de datos a dos carácteres o aumentar la frecuencia de consulta a períodos de 100ms.
  b. La velocidad media es de 10cps pero la frecuencia máxima es de 60ms. La frecuencia de escaneo de la CPU tiene que ser como mínimo de 1/60ms → 16.66Hz

- 7.10 Consider a system employing interrupt-driven I/O for a particular device that transfers data at an average of 8 KB/s on a continuous basis.

  a. Assume that interrupt processing takes about 100 us (i.e., the time to jump to the interrupt service routine (ISR), execute it, and return to the main program). Determine what fraction of processor time is consumed by this I/O device if it interrupts for every byte.
  b. Now assume that the device has two 16-byte buffers and interrupts the processor when one of the buffers is full. Naturally, interrupt processing takes longer, because the ISR must transfer 16 bytes.While executing the ISR, the processor takes about 8 us for the transfer of each byte. Determine what fraction of processor time is consumed by this I/O device in this case.
  c. Now assume that the processor is equipped with a block transfer I/O instruction such as that found on the Z8000.This permits the associated ISR to transfer each byte of a block in only 2 us. Determine what fraction of processor time is consumed by this I/O device in this case.

- Desarrollo:

  a. 8kB/s → T_int_rq=1/8KB/s=125us → fracción=100us/125us=80 %
  b. T_interrupt_service=100us(ISR+1byte)+15bytesx8us=220us → T_16=16x125us=2ms → fracción=220us/2000us=0.11=11 %
  c. T_int_serv=100us(ISR+1byte)+15bytesx2us=130us → fracción=130us/2000us=6.5 %

- 7.11 In virtually all systems that include DMA modules, DMA access to main memory is given higher priority than CPU access to main memory. Why?

  - Si el buffer de datos del DMAC se llena y no es leído, se perderían los datos.

- 7.12 A DMA module is transferring characters to memory using cycle stealing, from a device transmitting at 9600 bps. The processor is fetching instructions at the rate of 1 million instructions per second (1 MIPS). Suponer que la CPU está continuamente capturando instrucciones (no captura datos). By how much will the processor be slowed down due to the DMA activity?

  - 1MIPS → 1 instrucción cada microsegundo. Como la CPU está continuamente capturando instrucciones tendrá ocupado el bus durante 1 microsegundo para captar cada instrucción y completar el ciclo de instrucción, por lo que el ciclo del bus del sistema es 1us.

- 1 character = 8 bits
- 9600 bps → 1200 bytes/s → 1/1200 seg/byte =833us/byte → cada byte se transfiere por robo de ciclo. Se roba el bus del sistema cada 833us, es decir, cada 833 ciclos del bus del sistema.
- El bus del sistema lo tiene el DMAC durante un ciclo, es decir, 1 us.
- Cada 833 ciclos el DMAC roba 1 → 1/833 → 0.12 %
- 1MIPSx(1-0.0012)=998800 instrucciones por segundo.

- 7.13 Consider a system in which bus cycles takes 500 ns. Transfer of bus control in either direction, from processor to I/O device or viceversa, takes 250 ns. One of the I/O devices has a data transfer rate of 50 KB/s and employs DMA. Data are transferred one byte at a time.

  a. Suppose we employ DMA in a burst mode. That is, the DMA interface gains bus mastership prior to the start of a block transfer and maintains control of the bus until the whole block is transferred. For how long would the device tie up the bus when transferring a block of 128 bytes?

  b. Repeat the calculation for cycle-stealing mode.

- Desarrollo:

  - 500ns dura el ciclo del bus del sistema
  - T_{tx}=1/50KB=20us/B. El DMA según recibe el dato del periférico lo transfiere a la memoria principal, transfiriendo datos a través del bus del sistema a la misma velocidad del periférico. Sólo tiene sentido en periféricos de alta velocidad.

    a. Modo ráfaga: $T=t_{acceso\_bus}+t_{bus\_io\_transferencia\_bloque}+t_{liberar\_bus}$=250ns+128x20us+250ns=2560us

    b. Robo de ciclo $T=128x(t_{acceso\_bus}+t_{bus\_io\_transferencia\_byte}+t_{liberar\_bus})$=128x(250ns+20us+250ns)=128x20.5us=2624us

- 7.16 A DMA controller serves four receive-only telecommunication links (one per DMA channel) having a speed of 64 Kbps each.

  a. Would you operate the controller in burst mode or in cycle-stealing mode?

  b. What priority scheme would you employ for service of the DMA channels?

- Desarrollo:

  a. Ahora el DMAC tendrá cuatro bufferes de datos y podría acceder al bus de la misma forma que con uno. Debido a que los enlaces de telecomunicaciones ocupan el canal de forma continua (voz o datos), todo el tiempo que dura la comunicaión, el modo ráfaga ocuparía el bus el 100 % del tiempo. Por lo que seleccionamos el robo de ciclo.

  b. Prioridad entre 4 clientes: misma prioridad ya que tienen la misma velocidad. Si tuviesen diferentes velocidades, tendría mayor velocidad el más rápido, el de mayor tráfico.

- 7.17 A 32-bit computer has two selector channels and one multiplexor channel. Each selector channel supports two magnetic disk and two magnetic tape units. The multiplexor channel has two line printers, two card readers, and 10 VDT terminals connected to it. Assume the following transfer rates:

  - Disk drive 800 KBytes/s
  - Magnetic tape drive 200 KBytes/s
  - Line printer 6.6 KBytes/s
  - Card reader 1.2 KBytes/s
  - VDT 1 KBytes/s
  - Estimate the maximum aggregate I/O transfer rate in this system.

    a. Los dos canales selector tienen los mismos periféricos. Un canal selector está permanentemente asignado a sus periféricos y sólo puede dar servicio a uno de los periféricos asignados. El multiplexor en cambio da servicio a todos → Rate=800+800+2x6.6+2x1.2+10x1=1625.6KB/s

- 7.18 A computer consists of a processor and an *I/O device D* connected to *main memory M* via a shared bus with a data bus width of one word. The processor can execute a maximum of 10^6 instructions per second. An average instruction requires five machine cycles, three of which use the memory bus. A memory read or write operation uses one machine cycle. Suppose that the processor is continuously executing "background" programs that require 95 % of its instruction execution rate but not any I/O instructions, es decir, el 5 % son instrucciones I/O si utiliza mecanismo e/s por programa. Assume that one processor cycle equals one bus cycle. Now suppose the I/O device is to be used to transfer very large blocks of data between M and D.

  a. If programmed I/O is used and each one-word I/O transfer requires the processor to execute two instructions, estimate the maximum I/O data-transfer rate, in words per second, possible through D.

  b. Estimate the same rate if DMA is used.

- Desarrollo:

  a. Mecanismo E/S por programa

     I. La transferencia se realiza por programa y lo realiza la CPU. La transferencia de 1 palabra requiere la ejecución de dos instrucciones.

     II. 1 instrucción=3 ciclos máquina con el memory bus

     III. Como el ciclo de bus equivale a un ciclo máquina $\rightarrow$ 3 ciclos de bus con el memory bus $\rightarrow$ La transferencia de una palabra requiere 3 ciclos de bus$\rightarrow$ En cada ciclo de bus se transfiere un tercio de la palabra.

     IV. Los programas en background requieren el 95 % de instrucciones a la CPU, dejando el 5 % de instrucciones para I/O

     V. Del 5 % de instrucciones i/o el 2.5 % son transferencias ya que hacen falta dos instrucciones i/o por transferencia.

     VI. T_transfer(1word)= 0.025x10^6 $\text{instrucciones}_{io}$/seg= 25000words/seg

  b. DMA:

     I. Observamos el tiempo que la CPU no utiliza el bus del sistema= 5 % de instrucciones (5 ciclos por instrucción) MÁS el 95 % de instrucciones (2ciclos por instrucción).

     II. El 5 % de ejecución de CPU, la CPU esta libre :10^6(inst/seg)x0.05x5(ciclos/instr)= 250000 ciclos/seg de procesador que utiliza el DMA= 250000 ciclos/seg de i/o que utiliza el DMA

     III. El 95 % de ejecución de CPU, el DMA comparte bus del sistema=10^6x0.95x2 ciclos libres de los 5 ciclos= 1900000 ciclos/seg de cpu= 1900000 ciclos/seg i/o

     IV. Total=1900000+250000=2.150.000 ciclos/seg bus i/o

     V. Si en cada ciclo se puede realizar una transferencia, esa sería la velocidad máxima. La CPU no realiza la operación de acceso a memoria, la realiza el controlador de memoria.

11. Capitulo 8: Operating System

- 8.3 A program computes the row sums Ci=Sum[aij] para j=1,n of an array A that is 100 by 100. Assume that the computer uses demand paging with a page size of 1000 words, and that the amount of main memory allotted for data is five page frames. Is there any difference in the page fault rate if A were stored in virtual memory by rows or columns? Explain.
  - Matriz A = 100x100 palabras = 10000 palabras
  - Memoria: 5 marcos de páginas : 5000 palabras
  - Proceso: 10000 palabras se dividirá en 10000/1000=10 páginas
  - Almacenamiento por *filas*
  - 1ª página: a1_1,a1_2,..,a1_100,a2_1,..,a2_100,..,..,a10_1,..,a10_100 → diez filas
  - 5ª página: a41_1,..,..,a50_100
  - 10ª página: a91_1 ,..,..,a100_100
  - xª página: desde a10*(x-1)+1_1 hasta a10*x_1 → diez filas
  - Ejecución primera fila: C1=SUM[a1j] j=1,100
    - Demand Paging:
      - ◇ La MP está vacía, ningún marco de página inicializado, todas las páginas en disco, sin copia en los marcos de la MPrincipal.
      - ◇ captura de a11 → FAULT (no está en MP, está en disco)→ copia 1ª página → obtiene C1
  - Ejecución C2 → a21 sí está en la primera página → obtiene C2
  - Ejecución C3,..,C10 → ningún fault ya que están en la primera página
  - Ejecución C11 → FAULT → copio la 2ª página
  - Ejecución C21 → FAULT → copio la 3ª página
  - FAULTS: C1,C11,C21,..,C91
  - Cada vez que se ejecutan las 100 filas Ci se producen 10 Fallos
  - El resultado hubiese sido el mismo si en lugar de 5 páginas hubiese tenido una página si la política de reemplazo es la FIFO
  - se podrían utilizar 4 marcos de página con los mismos datos y realizar los reemplazos en el mismo marco.
- Almacenamiento por *columnas* :
  - 1ª página: a1_1,a2_1,..,a100_1,a1_2,..,a100_2,..,..,a1_10,..,a100_10 → diez columnas
  - xª página: desde a1_10*(x-1)+1 hasta a100_10*x → diez columnas
    - Ejecución 1ª fila: C1=SUM[a1j] j=1,100
      - ◇ necesito cargar las 100 columnas de la fila 1 → necesito 10 páginas con diez columnas por página → 10 FAULTS
    - Ejecución nª fila: se necesitan 100 columnas que están distribuidas por páginas de 10 en 10 columnas. → hacen falta 10 páginas → 10 FAULTS
  - Cada vez que se ejecutan las 100 filas Ci : 10 Faults por fila → 1000 FAULTS
- 8.4 Consider a fixed partitioning scheme with equal-size partitions of 2^{16} bytes and a total main memory size of 2^{24} bytes. A process table is maintained that includes a pointer to a partition for each resident process. How many bits are required for the pointer?
  - Las tablas de descriptores están formadas por el índice y el contenido que en este caso es un puntero.
  - $2^{24}/2^{16}=2^8$ particiones de la memoria
  - Tamaño de 2^{16} → direcciones que terminan en hexadecimal en 0000 → direcciones k*2^{16}→0xnn0000
  - puntero: solo es necesario guardar los dos dígitos nn de mayor peso → 8 bits
    - luego se desplazan 16 bits a a la izda para tener la dirección base

- 8.6 Suppose the page table for the process currently executing on the processor looks like the following. All numbers are decimal, everything is numbered starting from zero, and all addresses are memory byte addresses. The page size is 1024 bytes.

VM

Figura 4: VM

- .

  a. Describe exactly how, in general, a virtual address generated by the CPU is translated into a physical main memory address.
     - La dirección virtual esta formada por los campos (VPN,VPO) → (base,offset). Mediante la tabla de paginas virtuales traducimos VPN en PPN. La dirección física es el par (PPN,PPO) donde el offset PPO=VPO
     - Para qué este cacheada la página virtual en la tabla de páginas virtuales el bit de validación tiene que valer 1.

  b. What physical address, if any, would each of the following virtual addresses correspond to? (Do not try to handle any page faults, if any.)

     a. 1052
        - VPN=Mod{1052/1024}=1 → Valid Bit=1 → PPN=7
        - VPO=Rest{1052/1024}=28
        - Dirección física = 7*1024+28=7196

     b. 2221
        - VPN=Mod{2221/1024}=2 → Valid Bit=0
        - No hay copia de esa página por lo que no se puede realizar la traducción

     c. 5499
        - VPN=Mod{5499/1024}=5 → Valid Bit=1 → PPN=0
        - VPO=Rest{1052/1024}=379
        - Dirección física = 0+379=379

- 8.8 A process references five pages, A, B, C, D, and E, in the following order: A; B; C; D; A; B; E; A; B; C; D; E .Assume that the replacement algorithm is first-in-first-out and find the number of page transfers during this sequence of references starting with an empty main memory with three page frames. Repeat for four page frames.

  a. MP → 3 marcos de página ; política FIFO
     - v/v/v; A→ A/v/v ; B→ A/B/v ; C→ A/B/C; D→ D/B/C; A→ D/A/C; B→ D/A/B; E→ E/A/B ; A→ E/A/B; B→ E/A/B; C→ E/C/B; D→ E/C/D; E→ E/C/D
     - 10 Fallos

  b. MP → 4 marcos de página ; política FIFO
     - v/v/v/v; A→ A/v/v/v ; B→ A/B/v/v ; C→ A/B/C/v; D→A/B/C/D; A→A/B/C/D; B→A/B/C/D; E→E/B/C/D; A→E/A/C/D; B→E/A/B/D; C→E/A/B/C; D→D/A/B/C; E→D/E/B/C

- 8.9 The following sequence of virtual page numbers is encountered in the course of execution on a computer with virtual memory: 3 4 2 6 4 7 1 3 2 6 3 5 1 2 3 Assume that a least recently used page replacement policy is adopted. Plot a graph of page hit ratio (fraction of page references in which the page is in main memory) as a function of main-memory page capacity n for 1⇐ n⇐8. Assume that main memory is initially empty.

Nº marcos	Fracción Aciertos	3 4 2 6 4 7 1 3 2 6 3 5 1 2 3
1	0	
2	0	
3	2/15	4 4 3 3
4	3/15	4 4 3 3 3
5	4/15	4 4 3 2 3 2 3
6	7/15	4 2 6 4 1 3 2 6 3 1 2 3
7	8/15	3 4 2 6 4 1 3 2 6 3 1 2 3
8	8/15	3 4 2 6 4 1 3 2 6 3 1 2 3

- 8.11 Suppose the program statement

```
for (i = 1; i <= n; i++)
  a[i] = b[i] + c[i];
```

- is executed in a memory with page size of 1000 words. Let n = 1000. Using a machine that has a full range of register-to-register instructions and employs index registers, write a hypothetical program to implement the foregoing statement. Then show the sequence of page references during execution.
  - En la memoria virtual estará tanto el código como los datos
  - Marcos de 1000 palabras.
  - Programa arquitectura load/store (espacio de direcciones virtual)

```
            SECCION CODIGO
            Ri <- 1
            Ra <- n
loop_start: R1 <- b[Ri]
            R2 <- c[Ri]
            R3 <- R1+R2
            a[Ri] <- R3
            Flags <- Ri<Ra
            Flags:PC <- loop_start
            CPU <- halt
            SECCION DATOS INICIALIZADOS
uno:        1
n:          1000
a:          array a[1000]
b:          array b[1000]
c:          array c[1000]
```

  - Asignación del espacio virtual
    - Código en la página PV1
    - array A ocupa una página → PV2
    - array B ocupa una página → PV3
    - array C ocupa una página → PV4
    - uno y n en una página de tipo datos → PV5
  - Ejecución
    - $1515(131411211)^{1000}11$

- 8.13 Consider a computer system with both segmentation and paging. When a segment is in memory, some words are wasted on the last page. In addition, for a segment size s and a page size p, there are s/p page table entries. The smaller the page size, the less waste in the last page of the segment, but the larger the page table. What page size minimizes the total overhead?

- Desarrollo:
  a. Número de páginas por segmento: tamaño del segmento/ tamaño de página = s/p
  b. Cada segmento tiene su propia tabla de páginas
  c. Si reducimos el tamaño de página se reduce la fragmentación interna pero se incrementa el número de entradas de la tabla de páginas.
  d. El Total de palabras desperdiciadas (w) es el desperdicio debido a las últimas páginas de cada segmento más el tamaño de la tabla de páginas. El valor medio de la fragmentación interna de todos los segmentos es p/2 y el tamaño de la tabla de páginas es proporcional al número de entradas de la tabla s/p → w=p/2+s/p → dw/dp=1/2-s/p^2=0 → p^2=2s

- 8.14 A computer has a cache, main memory, and a disk used for virtual memory. If a referenced word is in the cache, 20 ns are required to access it. If it is in main memory but not in the cache, 60 ns are needed to load it into the cache, and then the reference is started again. If the word is not in main memory, 12 ms are required to fetch the word from disk, followed by 60 ns to copy it to the cache, and then the reference is started again. The cache hit ratio is 0.9 and the main-memory hit ratio is 0.6. What is the average time in ns required to access a referenced word on this system?

  - T=hit_cache*t_acc_ca+(1-hit_cache)**hit_main**(t_main_cache+t_acc_ca)+(1-hit_cache)**(1-hit_main)**(t_acc_disk_main+t_main_ca
    = 0.9*20+0.1*0.6*(60+20)+0.1*0.4*(12000000+60+20)=480us

- 8.15 Assume a task is divided into four equal-sized segments and that the system builds an eight-entry page descriptor table for each segment. Thus, the system has a combination of segmentation and paging. Assume also that the page size is 2 Kbytes.

  a. What is the maximum size of each segment?
  b. What is the maximum logical address space for the task?
  c. Assume that an element in physical location 00021ABC is accessed by this task. What is the format of the logical address that the task generates for it? What is the maximum physical address space for the system?
     - Desarrollo:
       A. Tabla de PAGINAS: 8 entradas : 8 paginas virtuales de 2KB → Segmento:8*2KB=16KB.
          a. No dice nada pero ... La tabla de SEGMENTOS tendrá una entrada por segmento. Cada entrada de segmento apuntará a una tabla de página diferente. Una tabla de página por segmento.
       B. Proceso: 4 segmentos → 4*16KB=64KB
       C. Dirección lógica → Formato (Segmento,Pagina,VPO) → (4seg,8pag,2KB)→ (2bits,3bits,11bits)→ Dirección lógica de 16 bits
          a. Dirección física → 00021ABC → 8 digitos hex → 32 bits → 4GB
          b. Marcos de página → 4GB/2KB → 2*2^{20} marcos
          c. 00021ABC → 0000-0000-0000-0010-0001-1010-1011-1100 → marco/offset → 21/11 → 00000000000000001000011 / 01010111100 → marco 67/ offset 700
       D. Traducción: El offset virtual y físico idénticos (11bits) → El segmento lógico (2bits) apunta a una tabla de páginas. La página virtual (3bits) es el offset de la tabla de páginas. Cada entrada de la tabla de páginas es un puntero a un marco de la memoria principal (una dirección base de 21 bits). Se añadir la pregunta de inventarse la tabla de descripción de segmentos y las cuatro tablas de páginas de cada segmento. En este ejercicio la dirección lógica tendrá el offset 01010111100 y los 5 bits del par seg/página no se pueden saber ya que haría falta saber en que tabla y posición está el puntero al marco 67.

- 8.16 Assume a microprocessor capable of accessing up to 2^{32} bytes of physical main memory. It implements one segmented logical address space of maximum size 2^{31} bytes. Each instruction contains the whole two-part address. External memory management units (MMUs) are used, whose management scheme assigns contiguous blocks of physical memory of fixed size 2^{22} bytes to segments. The starting physical address of a segment is always divisible by 1024. Show the detailed interconnection of the external mapping mechanism that converts logical addresses to physical addresses using the appropriate number of MMUs, and show the detailed internal structure of an MMU (assuming that each MMU contains a 128-entry directly mapped segment descriptor cache) and how each MMU is selected.

  - un espacio virtual segmentado de 2^{31} bytes: no es el espacio virtual de cada segmento sino el de todos los segmentos.
  - dirección lógica con dos partes → (segmento,offset)
  - MP: Espacio de 2^{32} bytes con Bloques de 2^{22} bytes contiguos para cada segmento

- offset de 22 bits
- $2^{31}/2^{22} = 2^9$ segmentos en espacio virtual → 9 bits para el segmento en la dirección virtual y una tabla de segmentos con 512 entradas
- Dirección lógica de 31 bits (9,22) → (seg,offset)

- MP: segmentos alineados en multiplos de 1K
  - segmento físico:los 10 bits de menor peso son cero y los 22 de mayor peso están en la *tabla de segmentos*.
  - Dirección física: segmento+offset
- MMU: *tabla de segmentos* de 128 entradas (2^7)
  a. no tenemos una tabla con 512 entradas sino cuatro tablas de 128 cada una.
  b. cantidad de MMUs: Si tenemos 2^9 segmentos en el espacio virtual y la MMU tiene una tabla de 2^7 necesitaremos 4 MMUs.
  c. Traducción: espacio lógico (9,22)(seg,offset) en una dirección segmento+offset de 32 bits.
     a. De los 9 bits de segmento virtual, dos bits seleccionaran la MMU y otros siete bits la entrada de la tabla de segmentos. (2,7,22)
     b. los 9 bits de segmento lógico son el índice de la tabla de segmentos que contiene los 22 bits altos de un segmento físico.
     c. El offset físico también tiene un tamaño de 22 bits
     d. dirección física: dirección base múltiplo de 1K más offset de 22bits.

- 8.17 Consider a paged logical address space (composed of 32 pages of 2 Kbytes each) mapped into a 1-Mbyte physical memory space.

  a. What is the format of the processor's logical address?
  b. What is the length and width of the page table (disregarding the "access rights" bits)?
  c. What is the effect on the page table if the physical memory space is reduced by half?
     - Desarrollo:
     - MP : 1MB con páginas de 2KB → $2^{20}/2^{11} = 2^9$ marcos de página
       A. VPN/OFFSET → VPN:32 páginas supone 2^5, 5 bits ; OFFSET:2KB supone 2^{11}, 11bits
       B. Tabla de páginas: longitud igual al número de paginas virtuales= 32 y anchura igual al puntero a uno de los 2^9 marcos, es decir, 9 bits.
       C. Si se reduce la MP a la mitad, se reduce el número de marcos a la mitad también → 2^8 marcos de página → anchura de 8 bits.

- Randal Capítulo 9: Figure 9.19 shows the formats of the virtual and physical addresses. Since each page is 26 =64 bytes, the low-order 6 bits of the virtual and physical addresses serve as the VPO and PPO respectively.The high-order 8 bits of the virtual address serve as the VPN. The high-order 6 bits of the physical address serve as the PPN. Figure 9.20 shows a snapshot of our little memory system, including the TLB (Figure 9.20(a)), a portion of the page table (Figure 9.20(b)), and the L1 cache (Figure 9.20(c)). Above the figures of the TLB and cache, we have also shown how the bits of the virtual and physical addresses are partitioned by the hardware as it accesses these devices.

./images/ejercicios/randal_9-19.png

./images/ejercicios/randal_9-20.png

- Given this initial setup, let's see what happens when the CPU executes a load instruction that reads the byte at address 0x03d4
- Solución
  - TLBI:0x03
  - TLBT:0x3
  - VPN:0x0f
  - VPO:0x14

- PPN=0x0D
- physical address=0x354
- CO=0x0
- CI=0x5
- CT=0x0D
- Data=0x36

12. Capitulo 12: Processor Structure and Function (Capitulo 14 en 9ªEd)

- 12.1 a. If the last operation performed on a computer with an 8-bit word was an addition in which the two operands were 00000010 and 00000011, what would be the value of the following flags?
  - Carry
  - Zero
  - Overflow → Número con signo
  - Sign
  - Even Parity → Paridad PAR
  - Half-Carry
  - Desarrollo
    - 0010+0011=0101 → No hay llevada en el MSB, el resultado no es cero, no hay overflow, positivo, número de unos PAR, no hay llevada en el bit de posición 3. Por lo que todos los flags desactivados excepto el de paridad par . El flag parity estará a 1.
- 12.3 A microprocessor is clocked at a rate of 5 GHz.
  a. How long is a clock cycle?
  b. What is the duration of a particular type of machine instruction consisting of three clock cycles?
  - Desarrollo
    - a. T= 1/f = 1/(5*10^9)=0.2ns
    - b. 3T= 3*0.2=0.6ns
- 12.4 A microprocessor provides an instruction capable of moving a string of bytes from one area of memory to another. The fetching and initial decoding of the instruction takes 10 clock cycles. Thereafter, it takes 15 clock cycles to transfer each byte. The microprocessor is clocked at a rate of 10 GHz.
  a. Determine the length of the instruction cycle for the case of a string of 64 bytes.
  b. What is the worst-case delay for acknowledging an interrupt if the instruction is noninterruptible?
  c. Repeat part (b) assuming the instruction can be interrupted at the beginning of each byte transfer
  - Desarrollo
    - a) IC=Instruction Cycle= FI+DI+CO+FO+EI+WO; FI+DI=10T ; CO+FO= 0 ; EI= 15T/byte ; WO=0 ; T=1/(10*10^9))=0.1 ns; IC=(10+15*64)*T=970*0.1= 97ns
    - b) Justo nada más empezar la instrucción quedaría todo el ciclo para poder atender a la interrupción: 97 ns.
    - c) Si se interrumpe antes de la primera transfer tardaría 10T como mucho, y si se interrumpe durante las transferencias sería 15T. Por lo que es caso peor sería 15T=15*0.1=1.5ns
- 12.5 The Intel 8088 consists of a bus interface unit (BIU) and an execution unit (EU), which form a 2-stage pipeline. The BIU fetches instructions into a 4-byte instruction queue. The BIU also participates in address calculations, fetches operands, and writes results in memory as requested by the EU. If no such requests are outstanding and the bus is free, the BIU fills any vacancies in the instruction queue. When the EU completes execution of an instruction, it passes any results to the BIU (destined for memory or I/O) and proceeds to the next instruction. wikipedia: the 8088 had an **8-bit** external data bus
  a. Suppose the tasks performed by the BIU and EU take about equal time. By what factor does pipelining improve the performance of the 8088? Ignore the effect of branch instructions.
  b. Repeat the calculation assuming that the EU takes twice as long as the BIU.
  - Desarrollo
    - 0. El micro 8088 tiene un bus de datos de 8bits. La unidad de ejecución comprende la ALU y los Registros. La BIU junto a la EU forman conjuntamente una CPU segmentada con dos unidades. La *prefetch instruction queue* es el buffer que almacena la siguiente instrucción a ejecutar.

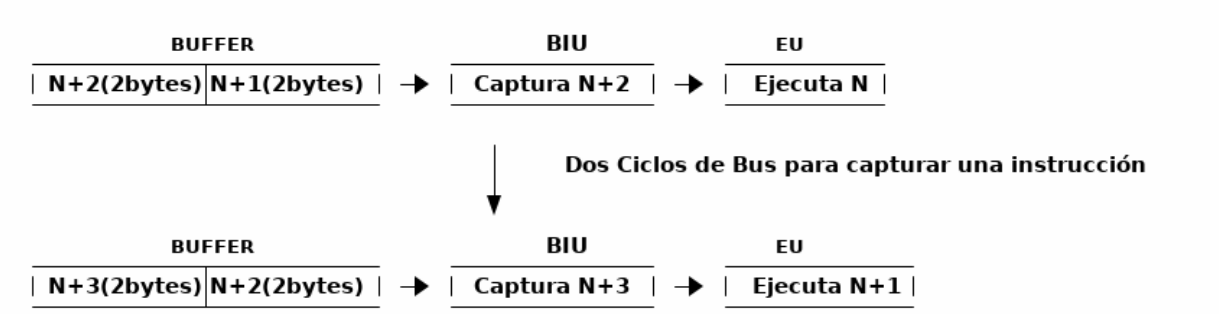
- a. Una etapa tarda x y la siguiente también x. La primera instrucción tarda en ejecutarse 2x y cada intervalo x sale una nueva por lo que la mejoría a partir de la segunda instrucción es de x/2x ⇒ En un ciclo de instrucción (duración 2x) salen instrucciones cada intervalo x, es decir, el doble.
- b. x+2x=3x. A partir de la segunda instrucción tardan 2x. En un ciclo de instrucción 3x salen instrucciones cada 2x ⇒ (3x time ciclo)/ (2x time/instrucción) = 1.5 veces más instrucciones por ciclo.

- 12.6 Assume an 8088 is executing a program in which the probability of a program jump is 0.1. For simplicity, assume that all instructions are 2 bytes long. If the prefetch instruction queue is empty, the EU waits for the next instruction byte to be fetched and shifted to top of the queue. When the EU executes a branch or jump instruction, it transfers control to a location corresponding to another set of sequential instructions. Whenever this happens, the BIU automatically resets the queue and then begins to fetch instructions from this new location to refill the queue.

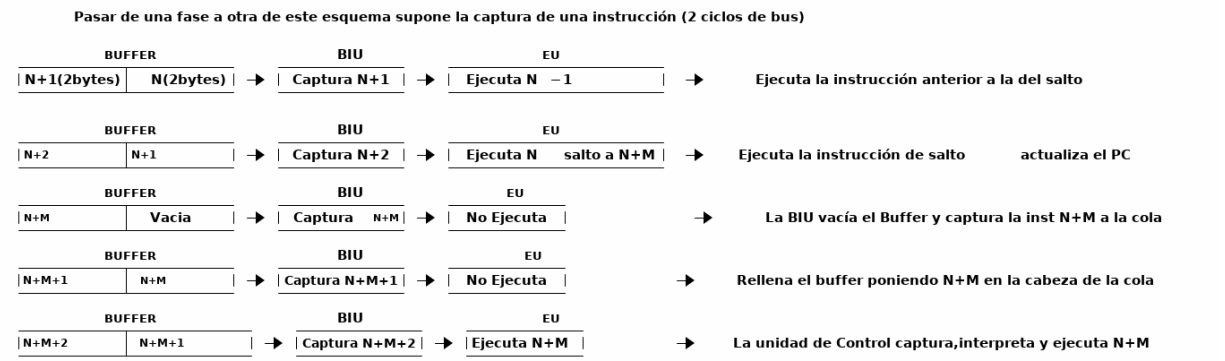
  a. What fraction of instruction fetch bus cycles is wasted?

  b. Repeat if the instruction queue is 8 bytes long.

  - Buffer (de 4 bytes para dos instrucciones) → BIU → EU : Para leer una instrucción son necesarios **DOS ciclos de bus**, un bus de datos (1 byte) por ciclo de bus.
  - Desarrollo
    - 0. Si no hay salto durante la ejecución de la instrucción N se captura la instrucción N+1.



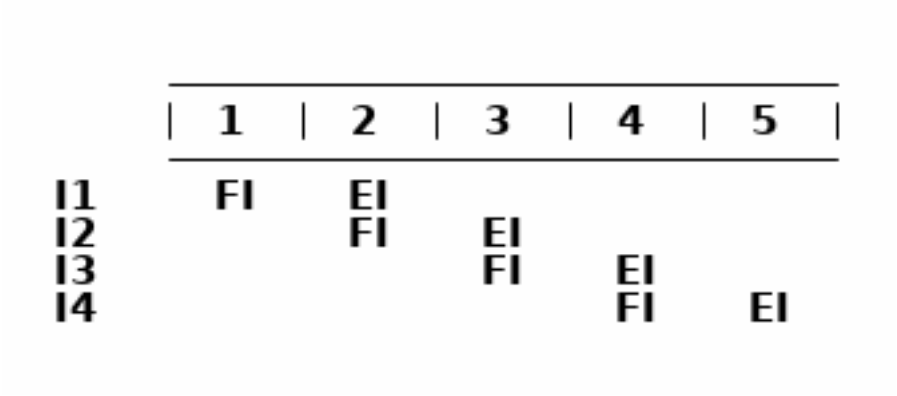
    - 1. Si la ejecución de la instrucción N supone un salto de M instrucciones no se ejecutará la instrucción N+1 que espera en el buffer, sino que se debe ejecutar la instrucción N+M. Durante la ejecución de N NO se captura nada sino que se actualiza el Contador de Programa a N+M y en el siguiente ciclo de instrucciónń se capturará N+M. Por lo que si hay salto, el ciclo de captura estará infrautilizado y será necesario vaciar el buffer de instrucciones.
    - Interpretación



    - a. Cuando la BIU capta de la memoria principal N+1 y lo pone en cola detrás de N, se están desaprovechando los dos ciclos de bus que se necesitan para la captura de la instrucción M+1 que no se va a ejecutar. Lo mismo ocurre con la captura de N+2 de la memoria princiapl. Por lo tanto se malgastan los ciclos del bus del sistema de N+1 y N+2, es decir, 4 ciclos de bus.
    - El buffer de instrucción es de 4 bytes según el ejercicio anterior, por lo que es necesario "empujar" los 4 bytes de N+1 y N+2 para dejar pasar a la nueva instrucción N+M desde que es capturada de la memoria principal.

- La captura de una instrucción no_jump supone 2 ciclos de bus bien utilizados, la instrucción estará en la cabeza del buffer cuando la vaya a ejecutar la CPU. La de una instrucción jump supone 2 ciclos bien utilizados en capturarla desde la memoria principal pero 4 ciclos mal utilizados en vaciar el buffer y desplazar la instrucción N+M desde la cola hasta la cabecera del buffer, mientras la cpu espera. De cada 100 instrucciones tendremos 100 instruccionesx2ciclos/instr bien utilizados y 10 instruccionesx4ciclos/instrucción mal utilizados→ en total 240 ciclos → fracción de infrautilización= 40/240= 0.166= 17 % del tiempo el bus no está siendo utilizado en operaciones fetch, la BIU está ocupado en vaciar el buffer.
- b. Con una cola de 8 bytes → Total=100x2+10*8=280→ ineficiencia=80/280=0.285=28.5 %

- 12.7 Consider the timing diagram of Figures 12.10. Assume that there is only a two-stage pipeline (fetch, execute). Redraw the diagram to show how many time units are now needed for four instructions.
  - Desarrollo



    - Son necesarias 5 unidades de Tiempo

- 12.9 A pipelined processor has a clock rate of 2.5 GHz and executes a program with 1.5 million instructions. The pipeline has five stages, and instructions are issued at a rate of one per clock cycle. Ignore penalties due to branch instructions and outof-sequence executions.
  a. What is the speedup of this processor for this program compared to a nonpipelined processor, making the same assumptions used in Section 12.4?
  b. What is throughput (in MIPS) of the pipelined processor?
    - Desarrollo
      - a. Duración Programa con N instrucciones, segmentación de k etapas de duración t cada una= 1ª instrucción más el resto = $k*t + (N-1)t = t(N+k-1) = para\ N>>k = t*(N-1)$. La relación sin_seg/con_seg = N*k*t / t(N+k-1) = Nk/(N+k-1) . Si N tiende a infinito → Nk/N=k=5
      - b. Throughput = instrucciones del programa/duración del programa= N/{t*(N+k-1)}=1/t

- 12.11 Consider an instruction sequence of length n that is streaming through the instruction pipeline. Let p be the probability of encountering a conditional or unconditional branch instruction, and let q be the probability that execution of a branch instruction I causes a jump to a nonconsecutive address. Assume that each such jump requires the pipeline to be cleared, destroying all ongoing instruction processing, when I emerges from the last stage. Revise Equations (12.1) and (12.2) to take these probabilities into account.
  - Desarrollo
    - Instrucciones cuya ejecución es un salto no consecutivo : pqn
    - Instrucciones cuya ejecución supone un no salto : (1-pq)n
    - T_programa=T_inst_salto+T_inst_nosalto = {pq*nkt}+{(1-pq)*(k+n-1)t}

- 12.13 Consider the state diagrams of Figure 12.28.
  a. Describe the behavior of each.

b. Compare these with the branch prediction state diagram in Section 12.4. Discuss the relative merits of each of the three approaches to branch prediction.

./images/ejercicios/cpu_12-13.png

- Predict taken: predicción de SI salto.
- Diagrama A:
  - Cambio de predicción afirmativa a negativa:
    - Partiendo de predicción de salto SI → Dos "NO" consecutivos para cambiar la predicción a NO
  - Cambio de predicción negativa a afirmativa:
    - Partiendo de predicción de salto NO → Un "SI" para cambiar la predicción a SÍ
- Diagrama B:
  - Cambio de predicción afirmativa a negativa:
    - Partiendo de predicción de salto SI → Dos "NO" consecutivos para cambiar la predicción a NO
  - Cambio de predicción negativa a afirmativa:
    - Partiendo de predicción de salto NO → Un "SI" para cambiar la predicción a SÍ si previamente ha habido dos "NO SALTO" consecutivos
    - Partiendo de predicción de salto NO → Dos "SI" para cambiar la predicción a SÍ si ha habido más de dos "NO SALTO" consecutivos
- 12.14 The Motorola 680x0 machines include the instruction *Decrement and Branch According to Condition*, which has the following form:

```
     DBcc Dn, displacement

where cc is one of the testable conditions, Dn is a general-purpose register, and  ↩
    displacement
specifies the target address relative to the current address.

The instruction can be defined as follows:
   if (cc = False)
     then begin
     Dn := (Dn) - 1;
     if Dn != -1 then PC := (PC) + displacement end
   else PC := (PC) + 2;
```

  - When the instruction is executed, the condition is first tested to determine whether the termination condition for the loop is satisfied. If so, no operation is performed and execution continues at the next instruction in sequence. If the condition is false, the specified data register is decremented and checked to see if it is less than zero. If it is less than zero, the loop is terminated and execution continues at the next instruction in sequence. Otherwise, the program branches to the specified location. Now consider the following assembly-language program fragment:

```
AGAIN CMPM.L (A0)+, (A1)+
      DBNE D1, AGAIN
      NOP
```

  - Two strings addressed by A0 and A1 are compared for equality; the string pointers are incremented with each reference. D1 initially contains the number of longwords (4 bytes) to be compared.
    a. The initial contents of the registers are A0 = $00004000, A1 = $00005000 and D1 = $000000FF (the $ indicates hexadecimal notation). Memory between $4000 and $6000 is loaded with words $AAAA. If the foregoing program is run, specify the number of times the DBNE loop is executed and the contents of the three registers when the NOP instruction is reached.

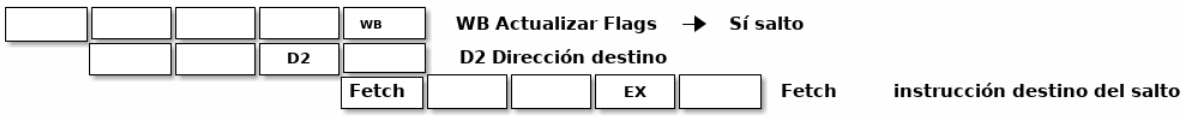
b. Repeat (a), but now assume that memory between $4000 and $4FEE is loaded with $0000 and between $5000 and $6000 is loaded with $AAA.

- Desarrollo:
  - La instrucción DBcc se emplea como control de los bucles una vez finalizada cada iteracción. La condicción cc hace referencia a la última operación antes de la instrucción DBcc, en este caso CMPM.L. Si la condición es verdadera → Fin de bucle y sigue la secuencia del programa. Si la condición es falsa decrementa el contador de iteraciones y salta al comienzo del bucle. Palabras tipo .L (Large) de 4 bytes. D1=0xFF. D1-1=0xFFFFFFFF. A0 puntero a string → (A0) indirección → (A0)+ incrementa el puntero en una palabra en cada ejecución.
    a. Los dos punteros apuntan a memoria cuyo contenido es $AAAA por lo tanto la comparación da como resultado EQUAL. La condición de la instrucción DBcc es NE, not equal, por lo tanto es FALSE y sí se ejecuta el bucle. Se ejecuta 0xFF+1 veces hasta llegar el contador D1=-1. Ultima dirección del puntero A0 → 0x4000+0xFFpalabras+1palabra=0x40 →2^2x(0xFF) equivale a desplazar 0xFF dos bits a la izda = 0x3FC → 0x4000+0x3FC+4=0x4400. Puntero A1 → 0x5000+0x3FC+4=0x5400.
    b. Todas las comparaciones dan como resultado distinto de cero → NE → por lo tanto TRUE → Unicamente se ejecuta una iteracción. D1=0xFF-1=0xFE; A0=A0+1palabra=0x4004 y A1=0x5004

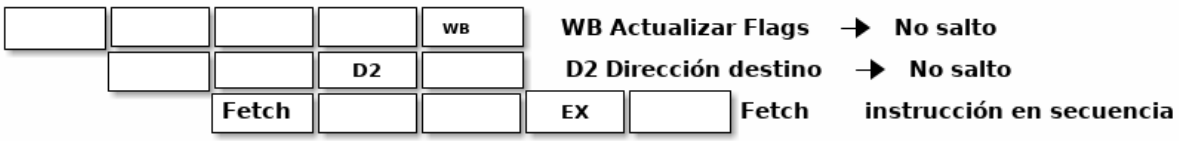
- 12.15 Redraw Figures 12.19c (14.21c), assuming that the conditional branch is not taken

  ./images/ejercicios/14-21.png

  - Desarrollo:
    - 80486: 32 bits
    - Etapas del cauce (pipeline) de instrucciones: FE-D1-D2-Ex-WB.
    - Fetch: Captura de la instrucción
    - D1: Decodifico Cod.Op y Modo Direccionamiento
    - D2: Operaciones para el cálculo de la Dirección Efectiva del Operando
    - EX: Operaciones ALU y acceso a operandos
    - WB: EFLAGS, Resultados en Reg y Mem(Caché y MP)
    - Figura 12.19 b) La 1ª ins. en EX lee el operando de la memoria y la 2ª en D2 accede a memoria para leer del puntero la dirección del operando.
    - Figura 12.19 c) La instrucción CMP actualiza reg flags. La instrucción Jcc en D2 ya tiene la dirección de salto aunque actualiza el Contador de Programa en EX. La inst 3ª después de D2 de Jcc ya pueded ser capturada.
    - Sí salto:
      - La cpu realiza la captura de la 3ª instrucción (Fetch) inmediatamente despúes de la captura de la segunda pero dicha captura es errónea ya que ha capturado la siguiente en secuencia a la 2ª y no la instrucción target. La captura de la instrucción destino se ha de realizar cuando se conozca la dirección dónde se encuentra dicha instrucción.



    - No salto:
      - La CPU captura las 3 instrucciones en secuencia y no se equivoca en la 3ª ya que no hay salto.



- 12.16 Table 14.5 summarizes statistics from [MACD84] concerning branch behavior for various classes of applications. With the exception of type 1 branch behavior, there is no noticeable difference among the application classes. Determine the fraction of all branches that go to the branch target address for the scientific environment. Repeat for commercial and systems environments.

  ./images/ejercicios/14-5.png

  - Desarrollo:
    - tipo1=72.5 ; tipo2=9.8 ; tipo3=17.7
      - ◇ tipo1: hay 3 casos dentro del tipo1 (1/3 salta incondicional,1/3 salta condicional,1/3 no salta)
      - ◇ tipo2: 91 %salta, el 9 % no salta
      - ◇ tipo3: saltan todas
    - Salto a destino= tipo1x[(0.2+0.4+0.35)x100/100+(43.2+24.3+32.5)x1/3]+ tipo2x0.91+ tipo3x100/100 =
    - Saltos to taget por aplicaciones
      - ◇ científica=tipo1x[(0.2)x100/100+(43.2)x1/3]+ tipo2x0.91+tipo3x100/100=0.724→ El 72 % de los saltos de una aplicación científica son a destino.
      - ◇ comercial =tipo1x[(0.4)x100/100+(24.3)x1/3]+ tipo2x0.91+tipo3x100/100=0.732
      - ◇ sistema =tipo1x[(0.35)x100/100+(32.5)x1/3]+ tipo2x0.91+tipo3x100/100=0.756

13. Capitulo 13: Reduces Instruction Set Computer (Capítulo 15 en 9ªEd)

- 13.3 We wish to determine the execution time for a given program using the various pipelining schemes discussed in Section 13.5. Let N = number of executed instructions, J = number of jump instructions, D = number of memory accesses. For the simple sequential scheme (Figure 13.6a) for a RISC architecture, the execution time is 2N+D stages. Derive formulas for two-stage, three-stage, and four-stage pipelining.

  ./images/ejercicios/15-6.png

  - Desarrollo:
    - CAUCE SEGMENTADO: I → captación de la instrucción. E→ operaciones ALU con Reg. u obtención de la dirección efectiva. D→ Transferencia Mem <→ Reg.
    - Cada instrucción tiene por lo menos dos etapas: E e I. En cambio la etapa D no la tienen todas las instrucciones (sólo load y store entre reg y mem)
      a. Figura apartado a →T=Nx(t_e+t_i)+t_dxD; si t_i=t_e=t_d=t → T=[2N+D]t
      b. Figura apartado b → Cauce segmentado → k=2 → Sin instrucciones de salto → $T_{k,n}$=[k+(n-1)]t → $T_{2,n}$=[2+(N-1)]t .
         a. Sólo es posible un acceso a memoria en cada etapa.
         b. I es una etapa, E y D forman una única etapa.
         c. E e I se solapan → N etapas E|I

            ```
            +-----+
            |  E  |
            +-----+
            |  I  |
            +-----+
            ```

         d. D no se solapa → D etapas
         e. En la fase E de la instrucción Branch se calcula la dirección de salto por lo que la fase I de la instrucción destino no puede coincidir con dicha fase E. Se soluciona con un instrucción de no operación NOOP.
         f. Los saltos originan un NOOP → una etapa de retardo más que añadir
         g. T=(N+D+J)t
      c. Figura apartado c → k=3 etapas
         a. En una etapa son posibles dos accesos a memoria.
         b. La 2ª instrucción load carga el dato en el registro en la etapa D por lo que no puede coincidir con la ejecución de la instrucción suma.

c. D,E e I se solapan si no hay dependencia de datos
d. Si hay dependencias D no se solapa por lo que hay una fracción de las D instrucciones que hay que sumar.

```
+-----+
|  E  |
+-----+
|  I  |
+-----+
                -> N+alfaxD
+-----+
|  E  |
+-----+
|  I  |
+-----+
|  D  |
+-----+
```

e. T=(N+alfa*D+J)t → J noops

d. Figura apartado d → k=4 etapas
a. Dividimos E en E1 (lectura RPG) y E2 (ALU y escritura RPG)

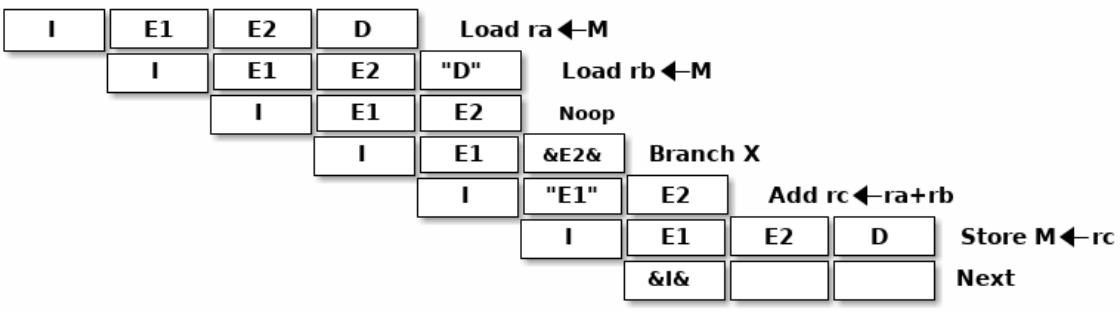
```
+-----+
|  D  |
+-----+
| E2  |
+-----+
| E1  |
+-----+
|  I  |
+-----+
```

b. El solape de D con dependencia de datos introduce un retardo y J otros dos según la figura.
c. T=(N+alfa*D+2J)t

- 13.4 Reorganize the code sequence in Figure 13.6d to reduce the number of NOOPs. Figura del ejercicio 13.3 d). risc_pipelining_13-6.png
  - La instrucción Branch la ejecutamos en la posición del 2º NOOP
  - Las dos instrucciones anteriores al salto (Add y Store) en lugar de los 2 NOOP después del salto Branch.

```
Load ra<-M
Load rb<-M
Noop
Branch X
Add  rc<-ra+rb
Store M<-rc
Next
```

    - Mientras se ejecutan Add y Store se calcula la dirección X



- Dependencias: Una vez que se carga rb en la fase "D" ya se puede leer rb en "E1"
- Dependencias: Una vez que se calcula X en &E2& ya se puede capturar la instrucción Next durante &I&

- 13.5 Consider the following code fragment in a high-level language:

```
for I in 1...100 loop
    S ← S + Q(I).VAL
end loop;
```

- Assume that Q is an array of 32-byte records and the VAL field is in the first 4 bytes of each record. Using x86 code, we can compile this program fragment as follows:

```
    MOV ECX,1 ;use register ECX to hold I
LP: IMUL EAX, ECX, 32 ;get offset in EAX
    MOV EBX, Q[EAX] ;load VAL field
    ADD S, EBX ;add to S
    INC ECX ;increment I
    CMP ECX, 101 :compare to 101
    JNE LP ;loop until I = 100
```

- This program makes use of the IMUL instruction, which multiplies the second operand by the immediate value in the third operand and places the result in the first operand (see Problem 10.13). A RISC advocate would like to demonstrate that a clever compiler can eliminate unnecessarily complex instructions such as IMUL. Provide the demonstration by rewriting the above x86 program without using the IMUL instruction.

  a. Array Q: 100 registros (estructuras) de 32 bytes cada uno.

  b. VAL field: primeros 4 bytes del registro.

  ```
  typedef struct {int VAL;....) Data;
  Data Q[100];
  ```

  c. Q(i).VAL : El campo VAL de cada registro Q(i)

  ```
      ## El bucle suma los campos VAL de los 100 registros
      MOV ECX,1 ;use register ECX to hold I #Indice de registro del array Q
  LP: IMUL EAX, ECX, 32 ;get offset in EAX  #EAX: dirección relativa del campo VAL del  ↩
      registro al que apunta el índice ECX del array Q
                                            #Cada 32 bytes un registro
      MOV EBX, Q[EAX] ;load VAL field       #Q en ensamblador se estructrua en bytes.  ↩
          100registrosx32bytes/registro=3200registros
      ADD S, EBX      ;add to S             #Suma de los 4 bytes del campo VAL
      INC ECX         ;increment I          #siguiente registro
      CMP ECX, 101    ;compare to 101       #ultimo+1 registro?
      JNE LP          ;loop until I = 100   #Siguiente interacción si no ultimo
  ```

  d. Multiplicar por 2^x equivale a un desplazamiento de x bits a la izda
  - Multiplicar por 32 → $x2^5$ → desplazar 5 bits a la izda : shl \$5, %ecx

- 13.6 Consider the following loop:

```
S := 0;
   for K := 1 to 100 do
S := S - K;
```

- A straightforward translation of this into a *generic* assembly language would look something like this:

```
   LD R1, 0          ;keep value of S in R1
   LD R2,1           ;keep value of K in R2
LP SUB R1, R1, R2    ;S := S - K
   BEQ R2, 100, EXIT ;done if K = 100          #Branch EQual
   ADD R2, R2, 1     ;else increment K
   JMP LP            ;back to start of loop
```

- A compiler for a *RISC* machine will *introduce* delay slots into this code so that the processor can employ the *delayed branch mechanism*.The JMP instruction is easy to deal with, because this instruction is always followed by the SUB instruction; therefore, we can simply place a copy of the SUB instruction in the delay slot after the JMP. The BEQ presents a difficulty. We can't leave the code as is, because the ADD instruction would then be executed one too many times. Therefore, a NOP instruction is needed. Show the resulting code.
- Desarrollo:

  a. Delayed Branch
  ./images/ejercicios/15-7.png
  - La primera gráfica es un salto normal y las otras dos retardado. La última gráfica supone una instrucción menos

  b. El programa presenta dos instrucciones de salto con retardo: BEQ y JMP

  c. JMP LP
  - Salto incondicional. En el programa original se ejecutará el salto después de `SUB R1, R1, R2`. Solución:

```
   JMP LP            ;back to start of loop
   ADD R2, R2, 1     ;else increment K
```

  - De esta manera el salto se ejecuta después del `ADD R2, R2, 1`

  d. BEQ R2, 100, EXIT
  - Salto condicional
  - Si lo dejamos como está el salto será después del ADD R2, R2, 1. Solución:

```
   BEQ R2, 100, EXIT ;done if K = 100          #Branch EQual
LP SUB R1, R1, R2    ;S := S - K
```

  - Ahora el salto condicional se realizará después de la resta SUB

  e. Solución 1ª:

```
   LD R1, 0          ;keep value of S in R1
   LD R2,1           ;keep value of K in R2
LP BEQ R2, 100, EXIT ;done if K = 100          #Branch EQual
   SUB R1, R1, R2    ;S := S - K
   JMP LP            ;back to start of loop
   ADD R2, R2, 1     ;else increment K
```

  - Tiene el defecto de que si R2=100 se ejecuta también SUB modificando el valor final de R1 y R2

  f. Solución Definitiva:

```
   LD R1, 0          ;keep value of S in R1
   LD R2,1           ;keep value of K in R2
LP BEQ R2, 100, EXIT ;done if K = 100          #Branch EQual
   NOP
   ADD R2, R2, 1     ;else increment K
   JMP LP            ;back to start of loop
   SUB R1, R1, R2    ;S := S - K
```

- 13.7 A RISC machine may do both a mapping of symbolic registers to actual registers and a *rearrangement* of instructions for pipeline efficiency. An interesting question arises as to the order in which these two operations should be done. Consider the following program fragment:

```
LD SR1,A          ;load A into symbolic register 1
LD SR2, B         ;load B into symbolic register 2
ADD SR3, SR1, SR2 ;add contents of SR1 and SR2 and store in SR3
LD SR4, C
LD SR5,D
ADD SR6, SR4, SR5
```

a. First do the register mapping and then any possible instruction reordering. How many machine registers are used? Has there been any pipeline improvement?
b. Starting with the original program, now do instruction reordering and then any possible mapping. How many machine registers are used? Has there been any pipeline improvement?

- Desarrollo: .
- 13.9 In many cases, common machine instructions that are not listed as part of the MIPS instruction set can be synthesized with a single MIPS instruction. Show this for the following:

a. Register-to-register move
b. Increment, decrement
c. Complement
d. Negate
e. Clear

- 13.11 SPARC is lacking a number of instructions commonly found on CISC machines. Some of these are easily simulated using either register R0, which is always set to 0, or a constant operand. These simulated instructions are called pseudoinstructions and are recognized by the SPARC compiler. Show how to simulate the following pseudoinstructions, each with a single SPARC instruction. In all of these, src and dst refer to registers. (Hint: A store to R0 has no effect.)

a. MOV src, dst
b. COMPARE src1, src2
c. TEST src1
d. NOT dst
e. NEG dst
f. INC dst
g. DEC dst
h. CLR dst
i. NOP

- 13.12 Consider the following code fragment:

```
if K > 10
  L := K + 1
else
  L := K - 1;
```

- A straightforward translation of this statement into SPARC assembler could take the following form:

```
    sethi %hi(K), %r8        ;load high-order 22 bits of address of location
                          ;K into register r8
    ld [%r8 + %lo(K)], %r8   ;load contents of location K into r8
    cmp %r8, 10              ;compare contents of r8 with 10
    ble L1                   ;branch if (r8) <= 10
    nop
    sethi %hi(K), %r9
    ld [%r9 + %lo(K)], %r9   ;load contents of location K into r9
    inc %r9                  ;add 1 to (r9)
    sethi %hi(L), %r10
    st %r9, [%r10 + %lo(L)]  ;store (r9) into location L
    b L2
    nop
L1: sethi %hi(K), %r11
    ld [%r11 + %lo(K)], %r12 ;load contents of location K into r12
    dec %r12                 ;subtract 1 from (r12)
    sethi %hi(L), %r13
    st %r12, [%r13 + %lo(L)] ;store (r12) into location L
L2:
```

- The code contains a nop after each branch instruction to permit delayed branch operation.

  a. Standard compiler optimizations that have nothing to do with RISC machines aregenerally effective in being able to perform two transformations on the foregoing code. Notice that two of the loads are unnecessary and that the two stores can be merged if the store is moved to a different place in the code. Show the programafter making these two changes.

  b. It is now possible to perform some optimizations peculiar to SPARC. The nop after the ble can be replaced by moving another instruction into that delay slot and setting the annul bit on the ble instruction (expressed as ble,a L1). Show the program after this change.

  c. There are now two unnecessary instructions. Remove these and show the resulting program