# IS09 - Introducción a los Computadores
# Examen Final - 09 de julio de 2002.

Ingeniería Técnica en Informática de Sistemas.
SOLUCIONES

## 1. Primer parcial

- Pregunta 1

  La arquitectura Von Neumann almacena tanto los datos como las instrucciones en la memoria del sistema, que es única –es decir, no hay memoria de datos y memoria de instrucciones separadas–. Esto podría afectar tanto a la organización del procesador básico como a las fases de ejecución de las instrucciones.

  En cuanto a la organización, habría que gestionar de qué forma se puede acceder a memoria con diferentes registros que indican la dirección del acceso –uno sería el contador de programa y otros los campos fuente y destino del registro de instrucciones. En cuanto a las fases, parece difícil que se puedan realizar dos accesos simultáneos a la memoria –requerido cuando hay que leer dos operandos fuente– por lo que cada acceso debería hacerse en una fase distinta.

- Pregunta 2

  La tabla siguiente muestra el contenido de los registros internos al ejecutar cada una de las fases de las instrucciones. Los valores que aparecen en itálica son los que se modifican en la fase en curso.

  Los encabezados de las columnas son: **F** = fase, **CP** = contador de programa, **Op.** = operación, **Im.** = inmediato, **F1** = fuente1, **F2** = fuente2, **D** = destino y **Sal.** = salida.

| **Instrucción** | **F** | **CP** | **Registro de Instrucciones** | | | | | | **UA1** | **UA2** | **Sal.** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | **Tipo** | **Op.** | **Im.** | **F1** | **F2** | **D** | | | |
| `movec d1, 0` | 0 | *1* | *movec* | | *0* | | | *d1* | 0 | 0 | 0 |
| | 1 | 1 | movec | | 0 | | | d1 | 0 | 0 | 0 |
| | 2 | 1 | movec | | 0 | | | d1 | 0 | 0 | 0 |
| | 3 | 1 | movec | | 0 | | | d1 | 0 | 0 | 0 |
| `resta d0, d1, d1` | 0 | *2* | *alu* | – | | *d1* | *d1* | *d0* | 0 | 0 | 0 |
| | 1 | 2 | alu | – | | d1 | d1 | d0 | *0* | *0* | 0 |
| | 2 | 2 | alu | – | | d1 | d1 | d0 | 0 | 0 | *0* |
| | 3 | 2 | alu | – | | d1 | d1 | d0 | 0 | 0 | 0 |
| `sumac d1, 1` | 0 | *3* | *aluc* | + | *1* | *d1* | | *d1* | 0 | 0 | 0 |
| | 1 | 3 | aluc | + | 1 | d1 | | d1 | *0* | *1* | 0 |
| | 2 | 3 | aluc | + | 1 | d1 | | d1 | 0 | 1 | *1* |
| | 3 | 3 | aluc | + | 1 | d1 | | d1 | 0 | 1 | 1 |
| `salta>d0, d1, 0` | 0 | *4* | *saltoc* | > | *0* | *d0* | *d1* | | 0 | 1 | 1 |
| | 1 | 4 | saltoc | > | 0 | d0 | d1 | | *0* | *1* | 1 |
| | 2 | 4 | saltoc | > | 0 | d0 | d1 | | 0 | 1 | *falso* |
| | 3 | 4 | saltoc | > | 0 | d0 | d1 | | 0 | 1 | `falso` |

- Pregunta 3

  La tabla siguiente muestra los números a convertir y su traducción en binario en las tres primeras columnas. La cuarta indica el valor en complemento a 2 con todos los bits necesarios. Se recuerda que los números negativos deben tener un 1 como bit más significativos, y los positivos un cero. La columna cinco indica el número mínimo de bits necesarios para representar cada número, y la siguiente los números representados con 10 bits, el valor mínimo capaz de representarlos a todos. Obsérvese cómo los números se extienden en signo –añadiendo ceros a la izquierda en los positivos y unos en los negativos– para representarlos en un formato mayor.

| | Decimal | **Binario natural** | **Ca2** | **bits** | **Ca2 con 10 bits** |
|---|---|---|---|---|---|
| A | -512 | 10 0000 0000 | 10 0000 0000 | 10 | 10 0000 0000 |
| B | -111 | 110 1111 | 1001 0001 | 8 | 11 1001 0001 |
| C | 143 | 1000 1111 | 0 1000 1111 | 9 | 00 1000 1111 |
| D | 308 | 1 0011 0100 | 01 0011 0100 | 10 | 01 0011 0100 |

  Si realizamos la suma A + B obtenemos 101 1001 0001, que en un formato de diez bits sería 01 1001 0001. Este valor representa un número positivo pues su bit más significativo es 0, por lo que el resultado de la suma es incorrecto en el formato de representación elegido –recuérdese que se han sumado dos números negativos.

  Por el contrario, la suma C + D resulta 01 1100 0011 que es correcto pues es suma de dos valores positivos.

- Pregunta 4

  El circuito es secuencial pues existe una realimentación de las salidas hacia las entradas de las puertas. En este caso además, se tiene a la entrada un biestable típico RS formado por las puertas NOR del primer nivel. En cualquier caso, es fácil demostrar –mediante tablas de transiciones y de verdad o ecuaciones lógicas– que la salida **S** del circuito siempre vale 0.

# 2. Segundo parcial

- Pregunta 1

  Los formatos de instrucción *a*, *b* y *c* de las figuras nos van a permitir responder de forma sencilla a los cuatro apartados.

  - El formato *a* utiliza 2 bits para el opcode. Los formatos *b* y *c* utilizan 5, pero en estos formatos no se pueden repetir las combinaciones de bits de las instrucciones que utilizan el formato *a*. De tal forma, con 2 bits tendremos $2^2 = 4$ combinaciones posibles. Dos de ellas las utiliza el primer formato, luego nos quedan otras 2 para los formatos *b* y *c*. Estos formatos tienen 3 bits adicionales para el código de operación, por lo que cada una de estas 2 combinaciones de bits podrá repetirse hasta en $2^3 = 8$ instrucciones distintas, es decir: tendremos 2 instrucciones del formato *a* y $2 \times 8 = 16$ para los formatos *b* y *c*, en total 18 instrucciones distintas.
  - Los campos que indican registros, llamados *reg1*, *reg2* o *reg3* tiene 3 bits, por lo que podemos designar hasta $2^3 = 8$ registros distintos, que serán los registros de propósito general del procesador.
  - Los formatos *b* y *c* permiten específicar hasta 3 operandos, luego este es el número máximo por instrucción.
  - No sabemos nada acerca de los modos de direccionamiento ni otras restricciones del conjunto de instrucciones, así pues si el número máximo de operandos por instrucción es de 3, nada impide suponer que todos ellos pueden residir en memoria.

- Pregunta 2

  - La dirección efectiva en el modo indirecto es el contenido del registro, en este caso r1, es decir 3054.
  - La dirección efectiva en el modo directo a memoria es la dirección de memoria dada, 45600.
  - La dirección efectiva en el modo indirecto con despazamiento es el contenido del registro base más el desplazamiento indicado. En este caso $3082 + -244 = 2838$.
  - La dirección efectiva en el modo con índice y desplazamiento es la suma del registro base, del registro índice y del desplazamiento, es decir $40 + 3000 + 64 = 3104$.

- Pregunta 3

  El código puede darse según los convenios de paso de parámetros vistos en las prácticas o según los explicados en clase. A continuación aparece según lo explicado en clase. No se realiza ninguna optimización, en algunos casos el código es deliveradamente ineficiente para que resulte más claro.

  La función sería llamada apilando, en este orden, los valores de `col`, `fil` y `valor`.

```
; Se utilizan los siguientes alias y macros:
; sp sería r30
; fp sería r29

; push rx (siendo rx cualquier registro) sería:
;   addi sp, sp, -4
;   sw   rx, 0(sp)
;
; pop  rx (siendo rx cualquier registro) sería:
;   lw   rx, 0(sp)
;   addi sp, sp, 4
;
; push var (siendo var cualquier variable en memoria escrita en la forma desp(rx))
;   lw   r28, var
;   addi sp, sp, -4
;   sw   r28, 0(sp)

; int rellena(int valor, int fil, int col)
          push r31            ; Código de entrada. Se apilan los registros r31 y
          push fp             ; fp y se crea el puntero a frame fp.
          or   fp, sp, r0
; int bien, imas, imenos, jmas, jmenos
; todas en memoria menos la variable bien
          addi sp, sp, -16    ; guardamos espacio para las variables locales.
; la pila queda como sigue:
;
;         |   col   |     fp + 16
;         |   fil   |     fp + 12
;         |  valor  |     fp + 8
;         | dir.ret.|
;         |  fp ant.| <-- fp
;         |  jmenos |     fp - 4
;         |   jmas  |     fp - 8
;         |  imenos |     fp - 12
;         |   imas  |     fp - 16
```

```
;
; se definen los siguientes alias:
;
;  col    =  16(fp)
;  fil    =  12(fp)
;  valor  =   8(fp)
;  jmenos =  -4(fp)
;  jmas   =  -8(fp)
;  imenos = -12(fp)
;  imas   = -16(fp)
;
; bien = verifica(fil, col)
          push col            ; llamamos a la función verifica.
          push fil            ; la variable bien queda en r1
          jl   verifica
          addi sp, sp, 8
; if(bien == 0) return -1
          bneq r1, r0, sigue  ; seguimos si bien no es 0
          addi r1, r0, -1     ; en otro caso, devolvemos -1
          beq  r0, r0, final
; else {
; imas = fil + 1
   sigue: lw   r1, fil
          addi r2, r1, 1
          sw   r2, imas
; imenos = fil - 1
          addi r2, r1, -1
          sw   r2, imenos
; jmas = col + 1
          lw   r1, col
          addi r2, r1, 1
          sw   r2, jmas
; jmenos = col - 1
          addi r2, r1, -1
          sw   r2, jmenos
; if (vector[fil][col] != valor) {
          lui   r20, HI(vector) ; Cargamos en r20 la parte alta de vector
          ori   r20, r0, LO(vector) ;luego la perte baja.
                                ; recordemos que vector es una dirección de
                                ; memoria constante
          lw   r1, fil          ; multiplicamos el indice de fila por el
          ori  r2, r0, MAXCOL   ; número de columnas
          mulu r1, r2
          mflo r1
          lw   r2, col
          addu  r1, r1, r2     ; y le sumamos el índice de columna.
          sll  r2, r1, 2      ; multiplicamos por el tamaño de datos
          addu  r20, r20, r2   ; que es 4 y sumamos a vector.
          lw   r1, 0(r20)     ; leemos el elemento y comparamos con valor
          lw   r2, valor
          beq  r1, r2, fin2
; vector[fil][col] = valor
          sw   r2, 0(r20)
```

```
; rellena(valor, fil, jmas)
          push jmas
          push fil
          push valor
          jl   rellena
          addi sp, sp, 12
; rellena(valor, fil, jmenos)
          push jmenos
          push fil
          push valor
          jl   rellena
          addi sp, sp, 12
; rellena(valor, imas, col)
          push col
          push imas
          push valor
          jl   rellena
          addi sp, sp, 12
; rellena(valor, imenos, col)
          push col
          push imenos
          push valor
          jl   rellena
          addi sp, sp, 12
; } return 1
    fin2: addi r1, r0, 1 ; ponemos el valor de vuelta a 1
; }
   final: addi sp, sp, 16 ; arreglamos la pila y volvemos
          pop  fp
          pop  r31
          jr   r31
```