Ejercicios de la parte teórica

Ejercicio Teo.1

Haced este ejercicio siguiendo la metodología de trabajo que os hemos propuesto. Tenéis una semana para hacer este ejercicio, y después os daremos la solución.

Este ejercicio no se evalúa, si no lo podéis acabar todo no pasa nada, es para trabajar los contenidos de la asignatura y para que podáis comentar con el resto de compañeros vuestros resultados y las dudas que os puedan surgir.

Una vez que os facilitamos las soluciones, comparadlas con las vuestras, y verificáis que lo entendéis todo. Usad el foro de la asignatura para resolver cualquier duda que os surja. Tenéis que tener en cuenta que las soluciones no son únicas.

Para facilitar el trabajo os damos resuelto el primer ejercicio que es muy parecido al que os proponemos hacer a vosotros. El ejercicio se enuncia a continuación y damos la respuesta en azul.

Revisad con atención los apartados siguientes del módulo 7 sobre arquitectura CISCA puesto que los ejercicios se plantean sobre esta arquitectura:

"Apartado 1: Organización del computador", mirad sobre todo la parte del procesador (1.1), y en especial el punto 1.1.1 sobre la organización de los registros incluidos los bits de resultado. Muy **importante** en este apartado el punto **Bit de transporte (C) y Bit de desbordamiento.**
Apartado 1.2: memoria principal, para ver la organización de la memoria en la arquitectura CISCA.
Revisar la **tabla** de ejemplo de **Bits de resultado** en el Módulo 7, apartados 2.3.2. Instrucciones aritméticas y 2.3.3. Instrucciones lógicas, con múltiples combinaciones de operaciones y resultados.

"Apartado 2: Juego de instrucciones", fijaos en los operandos (punto 2.1), modos de direccionamiento (punto 2.2) y en las diferentes instrucciones (punto 2.3) que aparecen en los ejercicios.
Para los últimos ejercicios estudiad el punto 2.4 donde se ve cómo se traducen diferentes estructuras de control de lenguaje C a lenguaje ensamblador CISCA.

Ejercicio Teo.1.1.1 (resuelto)

Para cada una de las siguientes preguntas, suponed que el estado inicial del computador (el valor que contienen los registros, posiciones de memoria y los bits de resultado) antes de la ejecución de los fragmentos de código es el siguiente:

- Registros: Ri= i*4, por ejemplo: R0=0, R1=4, R2=8, etc. (excepto el R15 o SP, que tiene el valor inicial 0).
- Memoria: M(i)=(i+16) para i = 0, 4, 8,… (excepto en las posiciones de memoria en las que se encuentra el código del programa). Ejemplo: M(00001000h)=00001010h, etc. La notación M(i) se refiere a la palabra de 4 bytes M(i)..M(i+3) en little endian.
- Bits de resultado del registro de estado inactivos: Z=0, S=0, C=0, V=0
- Registros especiales: El valor del PC y del SP no son necesarios por la resolución de este ejercicio.

¿Cuál será el estado del computador después de ejecutar cada una de las siguientes instrucciones y fragmentos de código?
Indicad sólo el contenido de los registros y posiciones de memoria que se hayan modificado a consecuencia de la ejecución del código. Indicad el valor final de todos los bits de resultado. En este ejercicio no es necesario saber lo que ocupa cada una de las instrucciones en lenguaje máquina, ni os pedimos el valor del PC al ejecutar el código y no se necesario saber su valor para hacer los ejercicios.

Pregunta a)

```
ADD R3, R6
```

Valores iniciales:
R3 = 3 * 4 = 12, R6 = 6 * 4 = 24

Operación:
R3 = R3+R6 = 12d + 24d = 36d,
Modifica los bits de resultado, pero en este caso los valores generados son los mismos que los valores originales (el cero)

Resultado final (sólo registros y posiciones de memoria que cambian) y bits de condición.
R3 = 36d, Z = 0, S = 0, C = 0, V = 0
Z=0, porque el resultado 12 + 24 es diferente de 0
S=0, porque el resultado de 32 bits es positivo, y no tiene signo
C=0, porque la operación 12 + 24 no genera transporte
V=0, porque la operación 12 + 24 no genera acarreo

Pregunta b)

```
ADD R6, [FFFFFFF8h]
```

Valores iniciales:
Dirección de memoria: FFFFFFF8h (32 bits),

[FFFFFFF8h] contenido de la dirección FFFFFFF8h = Memoria(FFFFFFF8h)= FFFFFFF8h+16d = FFFFFFF8h+10h = 100000008h
pero como que el valor necesita 33 bits para representarlo, y los registros tienen una medida de 32 bits, se descarta el bit más significativo, y el resultado es:
Memoria(FFFFFFF8h) = (FFFFFFF8h + 10h) módulo 100000000h = 100000008h módulo 100000000h = 00000008h = 8d

R6 = 6 * 4 = 24d

Operación:
R6 = R6 + Memoria(FFFFFFF8h) = 24d + 8d = 00000018h + 00000008h = 00000020h = 32d

Resultado:
R6 = 32d = 00000020h = 20h, Z = 0, S = 0, C = 0, V = 0
Z=0, porque el resultado 24 + 8 es diferente de 0
S=0, porque el resultado de 32 bits es positivo, y no tiene signo
C=0, porque la operación 24 + 8 no genera transporte
V=0, porque la operación 24 + 8 no genera acarreo

Pregunta c)

```
ADD R4, -2
```

Valores iniciales:
-2d = en complemento a 2 (Can2): FEh
Para hacer el Ca2 de (-2): Cogemos el valor binario positivo: 2d= 00000010b, lo negamos: 11111101b y sumamos 1: 111111101b + 1 = 11111110b = FEh = FFFFFFFEh

R4 = 4 * 4 = 16d = 0000000010h

Operación:
R4 = R4 + (-2) = 16d + (-2d) = 10h + FEh = 00000010h + FFFFFFFEh = 0000000Eh = 0Eh = 14d

Resultado:
R4 = 14d = 0000000Eh, Z = 0, S = 0, C = 1, V = 0
Z=0, porque el resultado 16d + (-2d) es diferente de 0
S=0, porque el resultado de 32 bits es positivo, y no tiene signo
C=1, porque la operación 16d + (-2d) genera transporte (nos traemos una).
V=0, porque la operación 16d + (-2d) no genera acarreo

Pregunta d)

Código:

```
                MOV    R2, 3
                MOV    R1, R2
    Loop:       DEC    R2
                JE     End_loop
                MUL    R1, R2
                JMP    Loop
    End_loop:   MOV    [100], R1
```

Desarrollo instrucción a instrucción:
MOV R2,3
3d=03h=00000003h, valor que queremos poner al registre R2.
R2 = 2 * 4 = 8d
Operación: R2 = 3d

MOV R1, R2
R2 = 3d (ahora ya no vale 2 * 4 = 8d puesto que hemos cambiado el valor a la instrucción anterior).
R1 = 1 * 4 = 4d = 0000000004h
Operación: R1 = 3d = 03h = 00000003h

Loop: (1)
R2 = R2 – 1 = 3 – 1 = 2; Z=0, C=0, S=0, V=0
JE end_loop FALSO (Z desactivado)
R1 = R1 * R2 = 3 * 2 = 6
JMP loop (siempre salta)

Loop: (2)
R2 = R2 – 1 = 2 – 1 = 1; Z=0, C=0, S=0, V=0
JE end_loop FALSO (Z desactivado)
R1 = R1 * R2 = 6 * 1 = 6
JMP loop (siempre salta)

Loop: (3)
R2 = R2 – 1 = 1 – 1 = 0; Z=1, C=0, S=0, V=0
JE end_loop **VERDAD** (Z activado)

end_loop:
M[100] = 6

Solución Final:
R1 = 6, R2 = 0, M[100] = 6
Z = 1, S = 0, C = 0, V = 0
(la última instrucción que ha modificado los bits de resultado ha sido DEC R2 en el Loop: (3)).

Ejercicio Teo.1.1.2 (resuelto)

Escribís un programa en lenguaje ensamblador CISCA que tenga la funcionalidad que se describe mediante un código C en cada una de las siguientes preguntas.

Pregunta a)

Código C:

```
if (A > B) then A-- else B=B+4;
B = B * A;
```

Código ensamblador:

```
            MOV   R0, [A]
if:         CMP   R0, [B]
            JLE   else
then:       DEC   R0
            JMP   endif
```

```
else:   ADD   [B], 4
endif:  MUL   [B], R0
        MOV   [A], R0
```

Suponed que el estado inicial del procesador es el mismo que en el ejercicio anterior y que las direcciones simbólicas A y B valen 00000020h y 00000200h respectivamente ([A] = [00000020h] y [B] = [00000200h]). Memoria: M(i)=(i+8) para i = 0, 4, 8,…
¿Cuál será el estado del computador después de ejecutar este fragmento de código?

Desarrollo instrucción a instrucción:
MOV R0, [A]: R0 = Memoria(A) = Memoria(20h) = 20h + 8 = 28h
Las instrucciones de transferencia no modifican los bits de resultado y quedan como estaban.

CMP R0, [B]: R0 - Memoria(B)= 28h - Memoria(200h) = 28h – (200h+8) = 28h-208h = -1E0h = FFFFFE20h (en Ca2)
Z=0, S=1, C=1, V=0 (El resultado genera signo y transporte, pero no acarreo).

JLE else: CIERTO (S=1 implica resultado negativo, Z desactivado implica que no es cero).
Saltamos a la etiqueta else: y ejecutamos la instrucción ADD [B],4
Las instrucciones de salto no modifican los bits de resultado y quedan como estaban.

else: ADD [B],4
Memoria(B) = Memoria(B) + 4 = Memoria(200h) + 4 = 208h + 4 = 20Ch
Memoria(200h) = 20Ch (se modifica el valor de esta posición de memoria)
Z=0, S=0, C=0, V=0 (No se activa ningún bit de resultado).

endif: MUL [B], R0:
Memoria(B)= Memoria(B) x R0= Memoria(200h) x 28h = 20Ch x 28h =
(20Ch = (2 x 256 + 0 x 16 + 12 x 1) = 524d) (28h = 2 x 16 + 8 x 1 = 40d)
= 524 x 40 = 20960d = 051E0h
Memoria(200h) = 051E0h (se modifica el valor de esta posición de memoria)
Z=0, S=0, C=0, V=0 (no se activa ningún bit de resultado).

MOV [A], R0
Memoria(A) = R0 = 28h ; Memoria(20h) = 28 h
Las instrucciones de transferencia no modifican los bits de resultado y quedan como estaban.

Solución Final:
R0 = 28h,
M[A] = M[20h] = 28h (como antes de ejecutar),
M[B] = M[200h] = 051E0h
Z = 0, S = 0, C = 0, V = 0 (cómo se han dejar los bits de resultado a la última instrucción que los ha modificado, en este caso el MUL [B], R0).

¿Cuanto tendría que valer [A] inicialmente para que se ejecutara la instrucción DEC R0?

[A] = 209h

CMP R0, [B]: R0 - Memoria(B)=209h - Memoria(200h)=209h – (200h+8)=209h-208h=1
Z=0, S=0, C=0, V=0 (no se activa ningún bit de resultado y el JLE no saltaría).

Pregunta b)

Código C:

```
sum = 0;
While (num > 0) {
        sum = sum + num;
        num--;
}
```

Código ensamblador:

```
                MOV   R1, 0
while:          CMP   [num], 0
                JLE   end_w
                ADD   R1, [num]
                DEC   [num]
                JMP   while
end_w:          MOV   [sum], R1
```

Suponed que el estado inicial del procesador es el mismo que en el ejercicio anterior y que las direcciones simbólicas num y sum valen 00000004h y 00000008h respectivamente ([num] = [00000004h] y [sum] = [00000008h]). Memoria: M(i)=(i+8) para a i = 0, 4, 8,…

¿Cuál será el estado del computador después de ejecutar este fragmento de código?

Desarrollo instrucción a instrucción:

MOV R1, 0: R1 = 0

Las instrucciones de transferencia no modifican los bits de resultado y quedan como estaban.

(Recordad que [num]=[00000004h]=4h + 8=12d y [B]=[00000008h]=8h + 8=16d).

Loop: (1)

while: CMP [num], 0: Memoria(num) – 0 = Memoria(4h) – 0 = 12 – 0 = 12
Z=0, S=0, C=0, V=0 (no se activa ningún bit de resultado).

JLE end_w: (no hay ningún bit de resultado activo y el JLE no saltaría).
Las instrucciones de salto no modifican los bits de resultado y quedan como estaban.

ADD R1, [num]: R1 = R1 + Memoria(num)= 0 + Memoria(4)= 0 + 12 = 12
R1=12; Z=0, S=0, C=0, V=0 (no se activa ningún bit de resultado).

DEC [num]: [num] = [num] – 1 = Memoria(num) - 1 = Memoria(4) - 1 = 12-1 = 11
[num] = 11; Z=0, S=0, C=0, V=0 (no se activa ningún bit de resultado).

JMP while: (salta siempre a la etiqueta especificada y se vuelve a ejecutar la instrucción que hay la etiqueta 'while' CMP [num], 0).

Loop: (2)

while: CMP [num], 0: Memoria(num) – 0 = Memoria(4h) – 0 = 11 – 0 = 10
Z=0, S=0, C=0, V=0 (no se activa ningún bit de resultado).

JLE end_w: (no hay ningún bit de resultado activo y el JLE no saltaría).
Las instrucciones de salto no modifican los bits de resultado y quedan como estaban.

ADD R1, [num]: R1 = R1 + Memoria(num)= 0 + Memoria(4)= 12 + 11 = 23
R1=23; Z=0, S=0, C=0, V=0 (no se activa ningún bit de resultado).

DEC [num]: [num] = [num] – 1 = Memoria(num) - 1 = Memoria(4) - 1 = 11-1 = 10
[num] = 10; Z=0, S=0, C=0, V=0 (no se activa ningún bit de resultado).

JMP while: (salta siempre a la etiqueta especificada y se vuelve a ejecutar la instrucción que hay la etiqueta 'while' CMP [num], 0).

...

A cada iteración del bucle cambian R1 y [num]. Haremos una lista con los valores que van tomando cada vez que se ejecuta la instrucción después de while:

Loop:	1,	2,	3,	4,	5,	6,	7,	8,	9,	10,	11,	12,	13
R1:	0,	12,	23,	33,	42,	50,	57,	63,	68,	72,	75,	77,	78
[num]:	12,	11,	10,	9,	8,	7,	6,	5,	4,	3,	2,	1,	0

Cuando empieza la iteración 13, la condición de salida del bucle cambia:

Loop: (13)
while: CMP [num], 0: Memoria(num) – 0 = Memoria(4h) – 0 = 0 – 0 = 0
Z=1, S=0, C=0, V=0 (se activa el bit de cero).

JLE end_w: (cómo hay el bit de cero activo, el JLE saltaría a end_w).
Las instrucciones de salto no modifican los bits de resultado y quedan como estaban.

end_w: MOV [sum], R1: [sum] = Memoria(sum) = Memoria(8h) = R1 = 78
Las instrucciones de transferencia no modifican los bits de resultado y quedan como estaban.

Solución Final:
R1 = 78 = (12+11+10+9+8+7+6+5+4+3+2+1)
M[num] = M[4h] = 0
M[sum] = M[8h] = 78
Z = 1, S = 0, C = 0, V = 0

Ejercicio Teo.1.2.1

Para cada una de las siguientes preguntas, suponed que el estado inicial del computador (el valor que contienen los registros, posiciones de memoria y los bits de resultado antes de la ejecución de los fragmentos de código de cada pregunta) es el siguiente:

- Registros: Ri= i*2, por ejemplo: R0=0, R1=2, R2=4, etc. (excepto el R15 o SP, que tiene el valor inicial 0)
- Memoria: M(i)=(i+8) para i = 0, 4, 8,… (excepto en las posiciones de memoria en las que se encuentra el código del programa). La notación M(i) se refiere a la palabra de 4 bytes M(i)..M(i+3) en little endian.
- Bits de resultado del registro de estado inactivos: Z=0, S=0, C=0, V=0
- Registros especiales: El valor del PC y del SP no son necesarios por la resolución de este ejercicio.

¿Cuál será el estado del computador después de ejecutar cada una de las siguientes instrucciones y fragmentos de código?

Indicad sólo el contenido de los registros y posiciones de memoria que se hayan modificado a consecuencia de la ejecución del código. Indicad el valor final de todos los bits de resultado. En este ejercicio no es necesario saber lo que ocupa cada una de las instrucciones en lenguaje máquina, ni os pedimos el valor del PC al ejecutar el código y no se necesario saber su valor para hacer los ejercicios.

Pregunta a)

```
ADD R5, R6
```

Respuesta...

Pregunta b)

```
SUB [A13F00FCh], R6
```

Respuesta...

Pregunta c)

```
AND R6, FFFFFFF8h
```

Respuesta...

Pregunta d)

```
MOV [256], 3ACE0h
```

Respuesta...

Pregunta e)

```
CMP R2, R7
```

Respuesta...

Ejercicio Teo.1.2.2

Escribir un programa en lenguaje ensamblador CISCA que tenga la funcionalidad que se describe en el texto en cada una de las siguientes preguntas.

Pregunta a)

Escribir en R1 un 1 si el contenido de R2 es más grande o igual que el de R3 y además el de R4 es más pequeño que el de R5. Si no pasa todo el anterior, escribir un 0.

Respuesta...

Pregunta b)

Escribir en R1 un 1 si se cumple una y sólo una de las dos condiciones siguientes:

- el contenido de R2 es más grande o igual que el de R3
- el contenido de R4 es más pequeño que el de R5.

En cualquiera otro caso escribir en R1 un 0.

Respuesta...

Ejercicio Teo.1.2.3

Escribir un programa en lenguaje ensamblador CISCA que tenga la funcionalidad que se describe mediante un código C en cada una de las siguientes preguntas.

Pregunta a)

Código C:

```
if (A > B)  C = C + 3 else C = C – 1;
```

Código ensamblador:

Respuesta...

Pregunta b)

Código C:

Cálculo del Máximo Común Divisor por el algoritmo de Euclides

```
while (A != B) {
  if (A > B)  A = A – B;
     else   B = B – A;
}
MCD = A;
```

Código ensamblador:

Respuesta...

Pregunta c)

Código C:

Cálculo del término 'n-éssimo' (Sn) de la sucesión de Fibonacci. S0=0, S1=1, Sn=Sn-1+Sn-2.
Sn: next; Sn-1: second; Sn-2: first.

```
i = 0;
do {

  if (n<2) then {
    next=n;
  }
  else {
    next   = first+second;
    first   = second;
    second = next;
  }

  i++;

} while (i < n);
```

Código ensamblador:

Respuesta...

Pregunta d)

Código C:

Cálculo de power = b^e (b^e)

```
power=1;
for (i=e; i>0;i--) {
   power = power * b;
}
```

Código ensamblador:

Respuesta...