PEC1

Estructura de computadores

Programa

2019 s2

Estudios de informática, multimedia y comunicación

UOC
Universitat Oberta de Catalunya

Presentación

La presente PEC1 contiene 4 preguntas y representa el 50% de la nota de la evaluación continua.

Como podréis ver, los ejercicios son muy parecidos a los que habéis hecho durante estos días, en los que además habéis podido dar las soluciones, comentarlas y plantear dudas en el foro. Esta PEC es **individual**, **evaluable** y por tanto no puede comentarse.

Competencias

Las competencias específicas que persigue la PEC1 son:

- [13] Capacidad para identificar los elementos de la estructura y los principios de funcionamiento de un ordenador.
- [14] Capacidad para analizar la arquitectura y organización de los sistemas y aplicaciones informáticos en red.
- [15] Conocer las tecnologías de comunicaciones actuales y emergentes y saberlas aplicar convenientemente para diseñar y desarrollar soluciones basadas en sistemas y tecnologías de la información.

Objetivos

Los objetivos de la siguiente PEC son:

- Conocer el juego de instrucciones de la máquina CISCA.
- Conocer los modos de direccionamiento de la máquina CISCA.
- Traducir pequeños programas a ensamblador.
- Comprender la codificación interna de las instrucciones de ensamblador.
- Describir las microoperaciones involucradas en la ejecución de una instrucción de ensamblador.

Enunciado

Responder cada pregunta o apartado en el recuadro correspondiente.

Recursos

Podéis consultar los recursos disponibles en el aula, pero no hacer uso del foro.

Criterios de valoración

La **puntuación** de cada pregunta y los **criterios de evaluación** los encontraréis a cada pregunta.

Formato y fecha de entrega

- La PEC1 podéis entregarla en el apartado de **entrega de actividades** con el nombre **apellido1_apellido2_nombre_PEC1 (pdf / odt / doc / docx).**
- La fecha **límite** de entrega es el **13/03/2020.**

Enunciado

Pregunta 1 (2 puntos)

Suponed que el estado inicial del computador (el valor que contienen los registros, posiciones de memoria y bits de resultado justo antes de comenzar la ejecución de cada fragmento de código, de cada apartado) es el siguiente:

R0= 000h R1= 100h R2= 200h R3= 300h R4= 400h	M(00000100h)=F0F0F0F0h M(00000200h)=0F0F0F0Fh M(00000300h)=11110000h M(00000400h)=00001111h M(00000500h)=10000500h M(00000600h)=80000500h

- Bits de resultado del registro de estado: Z=0, S=0, C=0, V=0
- Registros especiales: suponemos que el PC apunta al inicio del fragmento de código de cada apartado.

¿Cuál será el estado del computador después de ejecutar cada uno de los siguientes fragmentos de código? Indicad solamente el contenido (**en hexadecimal**) de los registros y posiciones de memoria que se hayan modificado como resultado de la ejecución del código. Indicad el valor final de todos los bits de resultado. (No os pedimos que indiquéis el valor del PC después de ejecutar el código y por eso no os hemos dado el valor inicial del PC, donde comienza cada fragmento de código).

Importante: dejad claramente indicado (**preferiblemente resaltado o en otro color**) **al final de cada apartado el valor final** de los registros y direcciones de memoria modificadas, así como los bits de estado.

a)

```
NOT   [R4]
ADD   [R4], R1
NOT   [R4]
```

[R4] = NOT [400h] = NOT 00001111h = **11110000h**
[R4] = [400h] + 100h = 11110000h + 100h = **11110100h**
[R4] = NOT [400h] = NOT (11110100h) = **00001011h**

Z = 0 S = 0 C = 0 V = 0

b)

```
MOV R3, [R2]
ADD R3, [R1]
ADD R3, 1
```

R3 = [200h] = **0F0F0F0Fh**
R3 = 0F0F0F0Fh + [100h] = 0F0F0F0Fh + F0F0F0F0h = **FFFFFFFFh**
R3 = FFFFFFFFh + 1 = **00000000h**

Z = 1 S = 0 C = 1 V = 0

c)

```
PLUS:TEST [R4],0
     JE   END
     SAR  [R4], 4
     ADD  R0, 10h
     JMP PLUS
END: SUB R0, [500h]
```

[R4] = TEST [R4], 0 = **00000000h**
R0 = 00000000H – 10000500H = **EFFFFB00h**

Z = 0 S = 1 C = 1 V = 0

Criterios de valoración. Los apartados a) y b) valen 0,5 puntos cada uno y el c) vale 1 punto. La valoración de cada apartado es del 100% si no hay ningún error en la solución del apartado, es del 50% si el contenido de las posiciones de memoria y registros modificados es correcto, pero hay algún error en el contenido de uno o varios de los bits de resultado, y es del 0% si hay algún error en alguna posición de memoria o registro modificado.

Pregunta 2 (2 puntos)

En el código de alto nivel `M` es una variable de tipo vector de 10 elementos. Cada elemento de la matriz es un entero de 32 bits. En el programa ensamblador la matriz se encuentra almacenada a partir de la dirección simbólica `M`, en posiciones consecutivas (`M[0]`, en la dirección simbólica `M`, el siguiente, `M[1]`, en la dirección `M+4`, etc.).

El programa compara el orden creciente del vector desde el primer elemento sin pasarse de la dimensión, intercambiando el primer valor que no cumple la condición.

Utilizaremos el registro R1 para implementar la variable de control "i" y R2 para "i+1"

```
 I= 0;
 WHILE (M[I] <= M[I+1] && (I<8) {
    I:= I+1;
 }
IF (M[I] > M[I+1]){ aux := M[I+1]; M[I+1]= M[I]; M[I] = aux }
```

Rellena los espacios de la propuesta de programa ensamblador que se muestra a continuación para conseguir el resultado deseado.

```
          MOV R1, 0
          MOV R2, 4
PLUS:     MOV R3, [M + R1]
          CMP R3,[M + R2]
          JG ENDW2
          CMP R1, 32
          JG ENDW1
          ADD R1, 4
          ADD R2, 4
          JMP PLUS
ENDW1:    CMP R3, [M + R2]
          JLE END
ENDW2:    MOV R4, [M+R2]
          MOV [M+R2], R3
          MOV [M+R1], R4
END:
```

Criterios de valoración.

2 puntos. Se pierden 0,5 puntos por cada instrucción incorrecta.

Pregunta 3 (3 puntos)

Traducid a lenguaje máquina el fragmento de código en lenguaje ensamblador que os proponemos

```
            MUL R1, [V]
LOOP:       CMP R1, [V+100h]
            JE END
            ADD R1, [200h]
            JL LOOP
END:
```

en la tabla inferior. Suponed que la primera instrucción del código se ensambla a partir de la dirección **0003FC00h** (que es el valor del PC antes de empezar la ejecución del fragmento de código). Suponed que la dirección simbólica V vale **00404040h**. En la siguiente tabla usad una fila para codificar cada instrucción. Si suponemos que la instrucción comienza en la dirección @, el valor Bk de cada uno de los bytes de la instrucción con direcciones @+k para k=0, 1,… se debe indicar en la tabla en hexadecimal en la columna correspondiente (recordad que los campos que codifican un desplazamiento en 2 bytes o un inmediato o una dirección en 4 bytes lo hacen en formato little endian, esto hay que tenerlo en cuenta escribiendo los bytes de menor peso, de dirección más pequeña, a la izquierda y los de mayor peso, dirección mayor, a la derecha). Completad también la columna 'Dirección' que indica para cada fila la dirección de memoria del byte B0 de la instrucción que se codifica en esa fila de la tabla.

		Bk para k=0..10										
Dirección	**Ensamblador**	0	1	2	3	4	5	6	7	8	9	10
0003FC00h	MUL R1, [V]	22	11	20	40	40	40	00				
0003FC07h	CMP R1, [R2+100h]	26	11	52	00	01	00	00				
0003FC0Eh	JE END	41	60	0D	00							
0003FC12h	ADD R1, [200h]	20	11	00	00	02	00	00				
0003FC19h	JMP LOOP	40	00	07	FC	03	00					

Criterios de valoración. Cada instrucción ensamblada incorrectamente resta 0.5 puntos. Una instrucción está incorrectamente ensamblada si no se escribe el valor o se escribe un valor incorrecto de uno o varios de los dígitos hexadecimal que codifican la instrucción en lenguaje máquina.

Además, se resta 0.5 puntos si hay algún error en la columna que indica las direcciones de memoria en las que comienza cada instrucción.

Pregunta 4 (2 puntos)

El *ciclo de ejecución* de una instrucción se divide en 3 fases principales
1) Lectura de la instrucción
2) Lectura de los operandos fuente
3) Ejecución de la instrucción y almacenamiento del operando destino

Dar la secuencia de micro-operaciones que hay que ejecutar en cada fase para las siguientes instrucciones del código codificado en la pregunta anterior.

MUL R1, [V]

Fase	Micro-operaciones
1	MAR ←0003FC00h, READ ; Contenido del PC en el registro MAR MBR ←00404040201122h ; Leemos la instrucción PC ← 0003FC07h ; Incrementamos el PC en 7 unidades IR ←MBR ; Cargamos la instrucción en el registro IR
2	MAR ← Contenido IR(V), read MBR ← memoria
3	R1 ←R1 * MBR

JMP LOOP

Fase	Micro-operaciones
1	MAR ← 0003FC19h, READ ; Contenido del PC en el registro MAR MBR ← 0003FC070040h ; Leemos la instrucción PC ← 0003FC1Fh ; Incrementamos el PC en 6 unidades IR ← MBR ; Cargamos la instrucción en el registro IR
2	El operando fuente es un inmediato
3	PC ← 0003FC07h ; Actualizamos el Pc con la dirección de salto

Criterios de valoración. Si todo está correcto se obtienen 2 puntos. Cada instrucción es independiente y vale 1 punto. Se resta 0.5 por cada fallo dentro de la misma instrucción.

Pregunta 5 (1 punto)

Responde a las siguientes preguntas:

Pregunta 1. ¿Qué es el borrow y cómo se calcula?

Llamamos borrow al bit de transporte que se activa en caso de que al final de una resta nos llevamos una. Si al final de la operación de resta no se ha producido acarreo, el valor del bit de transporte será 0.

Pregunta 2. ¿En qué consiste la segmentación de instrucciones y cuál es su principal objetivo?

La técnica de segmentación de instrucciones consiste en dividir el ciclo de ejecución de las instrucciones en varias etapas que pueden coincidir o no con las fases del ciclo de ejecución de las instrucciones.

El principal objetivo es poder ejecutar simultáneamente las diferentes etapas de las diferentes instrucciones para aumentar el rendimiento del procesador.

Criterios de valoración. Cada pregunta individual vale 0,5 puntos si está correcta (respuesta razonada y que conteste a lo que se pregunta). 0 si no está correcta o es incompleta.