Examen 2019, preguntas y respuestas

Arquitectura de computadors i perifèrics (Universitat Autònoma de Barcelona)

PARCIAL 1 – ACiP

Problema 1 (3 puntos)

Traducir el código mostrado a continuación y escrito en lenguaje C a lenguaje ensamblador.
Consideraciones:

- Para las variables locales que son usadas sólo dentro del bucle, debéis utilizar registros para almacenar sus valores.
- Dependiendo del direccionamiento escogido, debéis utilizar los registros R0 y R1 (éste último si es necesario) para identificar la dirección del i-ésimo elemento del array *vector*, debéis utilizar el registro R2
- Hay que situar las instrucciones máquina a continuación de los comentarios identificados por las palabras TRADUCIR y SITUAR. Cualquier instrucción que no siga esta norma será considerada ERRONEA.
- No podéis utilizar la pila para alamacenar datos intermedios.

unsigned int vector [10];

```
::SITUAR:: la dirección inicial del array vector en el registro R0
        ADR R0, vector      //R0 contiene la dirección inicial del vector
for (int i=0; (i<8) && (vector[i]<100); i++)
{
::TRADUCIR:: int i=0
        MOV R1, 0       // R1 contiene el valor de la variable local i
::TRADUCIR:: Comparación del bucle for (i<8) && (vector[i]<100)
        inicio_for:
        CMP R1, 8       // Comparar la variable i con 8
        BGE salir       // Si es mayor o igual con signo, salir del bucle
        LDR R2, [R0]        // R2 contiene el dato i del array
        CMP R2, 100         // Comparar la variable del dato con 100
        BHS salir       // Si es mayor o igual sin signo, salir del bucle
        vector[i]=vector[i+1];
        ::TRADUCIR:: vector[i]=vector[i+1];
        ADD R2, R0, 4       // R2 contiene la dirección del siguiente elemento del array
        LDR R3, [R2]            // R3 contiene el dato del siguiente elemento del array
        STR R3, [R0]            // Guardar contenido de R3 en la dirección actual (i) de array
        MOV R0, R2              // R0 contiene la siguiente dirección del array
        ::TRADUCIR:: i++
        ADD R1, R1, 1
        B inicio_for
        }
        Salir:
```

OPCION 2:

::SITUAR:: la dirección inicial del array *vector* en el registro R0

```
ADR R0, vector      //R0 contiene la dirección inicial del vector
```

for (int i=0; (i<8) && (vector[i]<100); i++)

{

::TRADUCIR:: ***int i=0***

```
MOV R1, 0       // R1 contiene el desplazamiento entre las direcciones de los
diferentes elementos del vector, que como son enteros es de 4. És la variable i.
```

::TRADUCIR:: Comparación del bucle for (i<8) && (vector[i]<100)

```
inicio_for:
CMP R1, 40       // Al ser enteros el tamaño máximo del vector en bytes es 4*10=40
BGE salir       // Si es mayor o igual con signo, salir del bucle
LDR R2, [R0, R1]        // R2 contiene el dato i del array
CMP R2, 100        // Comparar la variable del dato con 100
BHS salir       // Si es mayor o igual sin signo, salir del bucle
```

vector[i]=vector[i+1];

```
::TRADUCIR:: vector[i]=vector[i+1];
ADD R2, R1, 4      // R2 contiene el desplazamiento al siguiente elemento del array, i+1
LDR R3, [R0, R2]          // R3 contiene el dato del siguiente elemento del array
STR R3, [R0, R1]          // Guardar contenido de R3 en la dirección actual (i) de array
::TRADUCIR:: i++
MOV R1, R2                // R1 ahora tiene el desplazamiento al siguiente elemento
B inicio_for
```

}

```
Salir:
```

PARCIAL 2018

Ejercicio 1

Considere la instrucción JUMP A10FF, R1 de salto incondicional a la dirección de memoria A10FF. Esta instrucción salva la dirección de retorno en el registro de propósito general R1.

a) Describa brevemente que hace esta instrucción con el contador de programa

Guardar el contenido de PC (dirección de retorno, PC tene la dirección de la siguiente instrucción) en el registro R1 y situar en el PC la dirección de memoria A10FF.

b) Desventajas que presenta utilizar registros generales para guardar la dirección de retorno

Solo permite un numero limitado (dependiendo del número de registros) de anidamientos de llamadas a función, por ese motivo se utiliza la pila, que permite almacenar un numero bastante mayor de direcciones de retorno.

Ejercicio 2

Dado un computador de 32 bits, que direcciona la memoria por bytes y con el siguiente formato de instrucción.

COD	REG	DIRECCION

10 bits – 6 bits – 16 bits

a) Si se utiliza direccionamiento directo, qué cantidad de memoria es posible direccionar con este formato de instrucción ?

2^16 = 64 kbytes

b) Si se utiliza direccionamiento indirecto, qué cantidad de memoria es posible direccionar con este formato de instrucción ?

2^32 = 4 Gbytes (computador de 32 bits = direcciones de 32 bits)

c) Si se utiliza direccionamiento indirecto relativo a registro base, qué cantidad de memoria es posible direccionar con este formato de instrucción ?

2^32 = 4 Gbytes (computador de 32 bits = direcciones de 32 bits)

Ejercicio 3

La siguiente figura presenta el esquema simplificado de un procesador de 32 bits. El procesador se encuentra conectado a una memoria que se direcciona por bytes y en cada acceso a memoria se leen o escriben 4 bytes. Los registros RT, TR1 y TR2 son registros temporales transparentes al usuario. El PC es un registro contador, con lo que no necesita pasar por ALU para actualizar su contenido.

(FIGURA)

Considere la siguiente sentencia de un lenguaje de alto nivel X = (A/C) + (B*D).

Suponiendo que las variables X, A, B, C, D residen en memoria, se pide:

A1) La secuencia de instrucciones en lenguaje ensamblador necesarias para ejecutar esta sentencia, suponiendo que el procesador dispone de los siguientes formatos para las instrucciones. (Registro a dirección // Registro a registro)

COD	REG	DIRECCION
COD	REG	REG

LOAD R0, A
DIV R0, C
LOAD R1, B
MUL R1, D
ADD R0, R1
STORE R0, X

A2) Cuantos accesos a memoria se realizan para los datos ?

5 accesos, ya que hay 5 instrucciones con 1 operando de memoria, con lo cual cada uno requiere un acceso

B1) La secuencia de instrucciones en lenguaje ensamblador necesarias para ejecutar esta secuencia con formato de insrtuccion dirección a dirección ?

DIV A, C
MUL B, D
ADD A, B
STORE A, X

B2) Cuantos accesos a memoria se realizan para los datos ¿

8 accesos, ya que todas las instrucciones tienen 2 operandos de memoria
+ 4 de escritura

C) Indicar las operaciones elementales correspondientes a la búsqueda y ejecución de la instrucción LOAD R1, A

BÚSQUEDA:
MAR <- PC
MBR <- { mem (MAR) }
IR <- MBR && PC <- PC + 4
(Decodificacion)

EJECUCIÓN (Búsqueda del operando):
MAR <- $IR_{dirección}$
MBR <- { mem (MAR) }
R1 <- MBR

(La instrucción ocupa 1 palabra de 32 bits)

Ejercicio 4

Traducir el código mostrado a continuación y escrito en lenguaje C a lenguaje ensamblador usado en las clases de problemas. Consideraciones:

- Para las variables usadas dentro del bucle, debéis usar registros para almacenar sus valores (el valor final se guarda al salir)
- Para los modos de direccionamiento que identifican las direcciones de los elementos de los arrays vector_a y vector_b solo podeis usar los registros R0, R1 y R2.
- Hay que situar las instrucciones maquina a continuación de los comentarios identificados por las palabras TRADUCIR y SITUAR. Cualquier instrucción que no siga esta norma será considerada ERRONEA.
- No podeis usar la pila.

Solucion 1

```
Int vector_a[16];
Int vector_b[16];
Unsigned int i;

::SITUAR:: La dirección inicial del array vector_a en el registro R0
        ADR R0, vector_a        // R0 contiene la dir inicial de vector_a
::SITUAR:: La dirección inicial del array vector_a en el registro R0
        ADR R1, vector_b        // R1 contiene la dir inicial de vector_b

I=0;
::TRADUCIR:: i=0;
        MOV R2, 0       //R2 contiene la variable local i. Despues se usara como despl.

While (i<16);
::TRADUCIR:: Comparacion del bucle while {i<16};
        Inicia_while:
        CMP R2, 64      // Compara R2 (despl) con 64 que es el tamaño max array 16*4
        BHS fuera_while         // Mayor o igual sin signo

If (vector_a[i] > 0) {
::TRADUCIR:: Comparacion del condicional {vector_a[i] > 0};
        LDR R3, [R0, R2]        //R3 contiene vector_a[i]
        CMP R3, 0
        BLE ir_else     // Menor o igual con signo

Vector_b[i] = vector_a[i] * vector_a[i];
::TRADUCIR:: Vector_b[i] = vector_a[i] * vector_a[i];
        MUL R3, R3, R3          // R3 contiene vector_a[i] * vector_a[i]
        STR R3, [R1, R2]        // Guardar R3 en vector_b[i]

Ir_else:

Else {
        Vector_b[i] = 0;
::TRADUCIR:: Vector_b[i] = 0
        MOV R3, #0
        STR R3, [R1, R2]        // Guardar R3 en vector_b[i]


(falta i++ cuidado con desplazamiento, B inicio_while y fuera_while)


Solucion 2:
```

```
Int vector_a[16];
Int vector_b[16];
Unsigned int i;

::SITUAR:: La dirección inicial del array vector_a en el registro R0
        ADR R0, vector_a        // R0 contiene la dir inicial de vector_a
::SITUAR:: La dirección inicial del array vector_a en el registro R0
        ADR R1, vector_b        // R1 contiene la dir inicial de vector_b

I=0;
::TRADUCIR:: i=0;
        MOV R2, 0       //R2 contiene el desplazamiento
        MOV R3, 0       // R3 contiene la variable local i

While (i<16);
::TRADUCIR:: Comparacion del bucle while {i<16};
        Inicia_while:
        CMP R3, 16      // Compara R3 (i) con 16
        BHS fuera_while         // Mayor o igual sin signo

If (vector_a[i] > 0) {
::TRADUCIR:: Comparacion del condicional {vector_a[i] > 0};
        LDR R4, [R0, R2]        //R4 contiene vector_a[i]
        CMP R4, 0
        BLE ir_else     // Menor o igual con signo

Vector_b[i] = vector_a[i] * vector_a[i];
::TRADUCIR:: Vector_b[i] = vector_a[i] * vector_a[i];
        MUL R4, R4, R4          // R4 contiene vector_a[i] * vector_a[i]
        STR R4, [R1, R2]        // Guardar R4 en vector_b[i]

Ir_else:

Else {
        Vector_b[i] = 0;
::TRADUCIR:: Vector_b[i] = 0
        MOV R4, #0
        STR R4, [R1, R2]        // Guardar R4 en vector_b[i]
} i++;
::TRADUCIR:: i++;
        ADD R2, R2, #4          // Desplazamiento al siguiente elemento de ambos array
        ADD R3, R3, #1          // Incrementa en 1 la variable i
        B inicio_while
Fuera_while:
        STR R2, i
```

1) RX ← [SP]
2) SP ← SP+4

DIR	INSTRUCCIÓN
40h	LDR R1, [R3]
44h	BL 80h
48h	STR R2,[R1,R3]

DIR	INSTRUCCIÓN
80h	PUSH R1
84h	BL 110h
88h	ADD R1, R2,R1
8Ch	POP R1
90h	RTS

DIR	INSTRUCCIÓN
110h	PUSH R2
114h	MUL R2, R3,R2
118h	POP R2
11Ch	RTS

		Direcciones de memoria (de 4 en 4)						Registros				
PC	Inst	0...3	4...7	8...11	12...15	16...19	20...23	R1	R2	R3	SP	DIR
		6	8	4	0	0	8	0	4	4	24	
40h	LDR R1, [R3]		read					8				4
44h	BL 80h						48h				24-4=20	20
80h	PUSH R1					8					20-4=16	16
84h	BL 110h				88h						16-4=12	12
110h	PUSH R2			4							12-4=8	8
114h	MUL R2, R3,R2								4*4=16			—
118h	POP R2			read					4		8+4=12	8
11Ch	RTS				read						12+4=16	12