Examen de Arquitectura de Computadoras

Febrero de 2021

Ejercicio 1 (se realiza en web)

Sea una CPU RISC de 16 bits con instrucciones de 16 bits y 8 registros de propósito general de 16 bits (Reg0-Reg7)

Instrucción	Descripción
LOAD REG1, REG2	Guarda en REG2 el contenido de la dirección de memoria almacenada en REG1
STORE REG1, REG2	Escribe el contenido de REG1 en la dirección de memoria almacenada en REG2.
NOP	No realiza ninguna operación
MOVI INM, REG	Carga el valor constante INM de 9 bits en REG (pone en 0 los 7 bits más significativos)
ADD REG1, REG2, REG3	Suma REG1 y REG2 y guarda el resultado en REG3
SUB REG1, REG2, REG3	Resta REG2 a REG1 y guarda el resultado en REG3
AND REG1, REG2, REG3	Realiza el and bit a bit entre REG1 y REG2 y guarda el resultado en REG3
OR REG1, REG2, REG3	Realiza el or bit a bit entre REG1 y REG2 y guarda el resultado en REG3
NOT REG1, REG2	Guarda en REG2 el resultado de realizar un NOT bit a bit al REG1
SL REG1, INM, REG2	Realiza el shift a la izquierda de tantos bits de REG1 como indique el inmediato INM (de 4 bits) y el resultado lo pone en REG2
SR REG1, INM, REG2	Realiza shift a la derecha de tantos bits de REG1 como indique el inmediato INM (de 4 bits) y el resultado lo pone en REG2.
CMP REG1, REG2	Instrucción de comparación que realiza la resta REG1-REG2 sin guardar el resultado, actualizando las banderas de condición.
JZ INM	Si la bandera Z está en 1 salta INM instrucciones desde la posición actual. INM es un número con signo de 12 bits
JN INM	Si la bandera N está en 1 salta INM instrucciones desde la posición actual. INM es un número con signo de 12 bits
JMP INM	Instrucción de salto incondicional, salta INM instrucciones desde la posición actual. INM es un número con signo de 12 bits

Se pide:

Diseñar el formato de instrucción presentado e indicar la codificación de cada instrucción, y compilar el siguiente fragmento de código utilizando la arquitectura anterior (no se acepta el uso de etiquetas)

```
short suma = 0;
for (short a = 1023; a < 4097; a++){
     suma = suma + a;
}
suma = suma / 2;
```

Solución:

Dado que tenemos 15 instrucciones precisamos 4 bits para codificarlas. Los registros son 8, por lo tanto se precisan 3 bits. Rellenaremos con 0 a la derecha los bits excedentes en caso necesario.

Utilizaremos los siguientes códigos y formatos:

```
LOAD      → 0000RRRrrr000000
STORE     → 0001RRRrrr000000
NOP       → 0010000000000000
MOVI      → 0011RRRIIIIIIII
ADD       → 0100RRRrrrDDD000
SUB       → 0101RRRrrrDDD000
AND       → 0110RRRrrrDDD000
OR        → 0111RRRrrrDDD000
NOT       → 1000RRRrrr000000
SL        → 1001RRRIIIIDDD00
SR        → 1010RRRIIIIDDD00
CMP       → 1011RRRrrr000000
JZ        → 1100IIIIIIIIIII
JN        → 1101IIIIIIIIIII
JMP       → 1110IIIIIIIIIII
```

La compilación del fragmento de código es la siguiente:

```
MOVI 0, R1      // R1 = suma
MOVI 256, R3
MOVI 1, R5
SL  R3, 2, R2  // R2 = a = 1024
SUB R2, R5, R2 // a = 1023
SL  R3, 4 R3   // R3 = 4096
ADD R3, R5, R3 // R3 = 4097

CMP R2, R3     // 4097 - a
JZ  4
ADD R1, R2, R1 // suma = suma + a
ADD R2, R5, R2 // a++
JMP -4

SR  R1, 1, R1
```

Ejercicio 2 (se realiza en web)

Sea el siguiente código que calcula la sumatoria de largos de los string de una lista de nodo

```
typedef struct{
     char *string;
     nodo *siguiente;
}nodo;

unsigned short largos(nodo *actual){
     if(actual == NULL)
          return 0;
     unsigned short largo = 0;
     while(actual->string[largo] != NULL)
          largo = largo + 1;
     return largo + largos(actual->siguiente);
}
```

Compile en 8086 sabiendo que el parámetro se recibe en stack y el resultado se devuelve en el stack. Todos los punteros se representan como desplazamientos respecto de DS. Calcule el consumo de stack para una lista de M nodos.

Solución:

```
largos proc
  POP AX ; dir ret
  POP SI ; SI=actual
  XOR DI, DI ; DI = largo
  CMP SI, 0
  JZ fin
  MOV BX, [SI] ; BX = string
while:
  CMP byte ptr [BX+DI], 0
  JZ finwhile
  INC DI
  JMP while
finwhile:
  PUSH AX ; guardo dir ret
  PUSH DI ; y largo actual
  PUSH [SI+2]
  CALL largos
  POP SI ; largos
  POP DI ; largo
  POP AX ; dir ret
  ADD DI, SI
fin:
  PUSH DI
  PUSH AX
  RET
largos endp
```

En esta implementación el paso base consume 4 bytes (2 para el parámetro y 2 para la dirección de retorno). El paso recursivo consume también 4 bytes (2 para guardar el largo calculado en el while y 2 para la dirección de retorno).
Por lo tanto para una lista de M nodos se consumen (M+1) * 4 bytes

Ejercicio 3 (se realiza en web)

Considere una CPU de 24 bits y direcciones de 24 bits, operando con una memoria cache y una memoria RAM direccionable de a bytes. La caché utiliza una función de correspondencia directa y tiene 128 líneas, cada una de 32 bytes. De todo esto se deduce que la dirección de memoria se interpreta como se indica a continuación:

[tag – 12 bits] [linea – 7 bits] [byte – 5 bits]

Considere la siguiente definición:

```
#define N …
typedef struct {
                        char dato;
   char otros_datos[N];
} mi_estructura;

mi_estructura[1024]  arreglo;
```

Considere el siguiente código:

```
int suma = 0;
for (int j = 0; j < 2; j++){
        for (int i = 0; i < 1024; i++){
           suma += arreglo[i].dato;
        }
}
```

Sabiendo que el arreglo se ubica a partir de la dirección 0 de memoria y que las variables i, j y suma se encuentran cargadas en registros, indique el hit rate en la ejecución de la sección de código indicada para los siguientes los casos de N = 3 y N = 127. Justifique sus respuestas.

Solución:

Lo primero a notar es que la caché tiene una capacidad de 4096 bytes.

Caso N = 3:

En este caso la estructura ocupa 4 bytes y por tanto el arreglo completo ocupa 4 * 1024 = 4096 bytes. Cada línea es capaz de guardar 8 elementos del arreglo. Como el arreglo no supera el tamaño total de la cache y el arreglo se dispone de forma contínua en memoria, y se encuentra alineada con la cahe, esta es capaz de contener al arreglo completamente. Teniendo esto en cuenta, el primer acceso a una línea nueva es miss, pero los siguientes 7 accesos son hit. Esto ocurre durante la primer recorrida del arreglo, dando 7/8 de hit rate en la primer iteración. En la siguiente iteración se obtienen todos hit ya que el arreglo está totalmente contenido en la cache (hit rate = 1 en la segunda iteración). Como la cantidad de accesos es la misma en ambas iteraciones, el hit rate total es el promedio de ambas: 15/16

(Una forma alternativa de cálculo es determinar el número de accesos totales y el número de hits totales y calcular #hits/#accesos)

Caso N = 127

Como los accesos son separados 128 bytes y el primero es a la línea 0, las direcciones accedidas son de la forma:

tag	línea	byte
0000000XXXXX	XXXXX00	00000

Los 7 bits menos significativos corresponden a la separación de 128 bytes entre accesos. Los bits marcados como X son los 10 bits que variarán al realiazar los 1024 accesos. De esto se puede ver, primero, que cada acceso es a una línea diferente, por lo que durante la primer iteración del for el hit rate = 0, y que, segundo, cada línea será utilizada 2^5 veces, lo que lleva a que cada línea será reemplazada 31 veces y entonces en cada acceso de la segunda iteración habrá un miss por conflicto.
Como consecuencia el Hit rate = 0

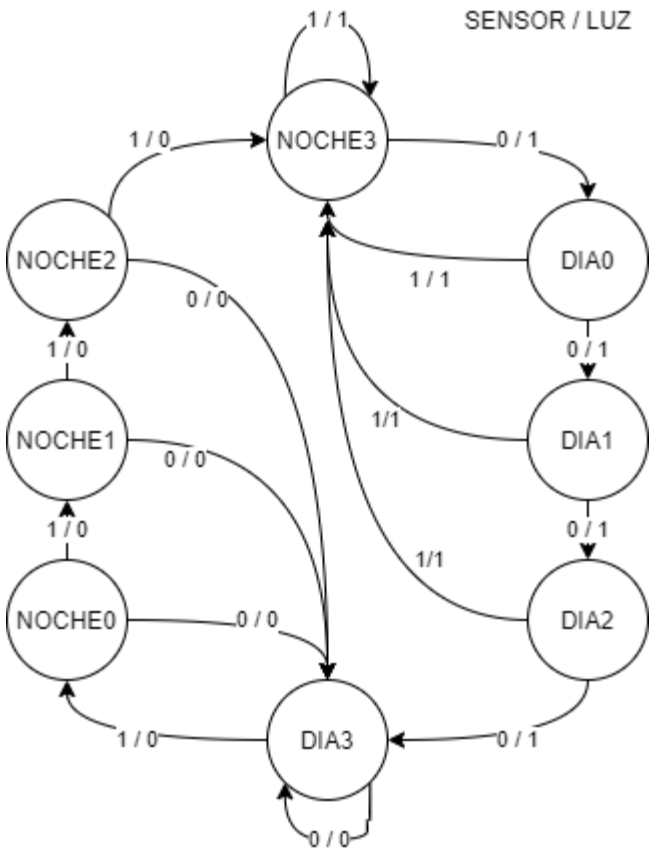
Ejercicio 4 (se realiza en papel, se entregan escaneos)

Se desea controlar una luz exterior de una casa de manera de encenderla en la noche y apagarla en el día. Para ello se dispone de un sensor de luz que determina si es de día o noche en función de la intensidad de la luz.

Para evitar prender y apagar la luz frente al ruido propio del sensor o de las condiciones ambientales deberán verificarse cuatro lecturas consecutivas de valor DIA para apagar la luz y cuatro lecturas de valor NOCHE para prender la luz.

Se pide:
Diseñar, utilizando la metodología del curso, un circuito que controle el encendido de la luz (un 1 enciende la luz) en base a la señal del sensor (0 = DIA, 1 = NOCHE), utilizando filp-flops tipo D y compuertas básicas. La ultima parte de la metodología (dibujar el circuito) no es requerida.

Solución:



Estado Actual	Entrada	Próximo Estado	Salida
NOCHE 0	0	DIA 3	0
NOCHE 0	1	NOCHE 1	0
NOCHE 1	0	DIA 3	0

NOCHE 1	1	NOCHE 2	0
NOCHE 2	0	DIA 3	0
NOCHE 2	1	NOCHE 3	0
NOCHE 3	0	DIA 0	1
NOCHE 3	1	NOCHE 3	1
DIA 0	0	DIA 1	1
DIA 0	1	NOCHE 3	1
DIA 1	0	DIA 2	1
DIA 1	1	NOCHE 3	1
DIA 2	0	DIA 3	1
DIA 2	1	NOCHE 3	1
DIA 3	0	DIA 3	0
DIA 3	1	NOCHE 0	0

Como se tienen 8 estados, se precisan 3 flip flops para guardarlo. La codificación de estados es la siguiente:

NOCHE 0: 000
NOCHE 1: 001
NOCHE 2: 010
NOCHE 3: 011
DIA 0: 100
DIA 1: 101
DIA 2: 110
DIA 3: 111

Tomando en cuenta la ecuación del flip flop tipo D: Q(n+1) = D(n), la tabla de verdad queda como se indica a continuación:

Estado Actual	**Entrada**	**Próximo Estado**	**Salida**
q2 q1 q0	Sensor	d2 d1 d0	Luz
000	0	111	0
000	1	001	0
001	0	111	0
001	1	010	0
010	0	111	0
010	1	011	0
011	0	100	1
011	1	011	1
100	0	101	1

100	1	011	1
101	0	110	1
101	1	011	1
110	0	111	1
110	1	011	1
111	0	111	0
111	1	000	0

Hallamos las expresiones mínimas utilizando Karnaugh:

q2q1\ q0s	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	1	0	0	1
10	1	0	0	1

d2 = !s

q2q1\ q0s	00	01	11	10
00	1	0	1	1
01	1	1	1	0
11	1	1	0	1
10	0	1	1	1

d1 = !q0 q1 + q2 !q1 s + q0 !s q2 + q0 s !q2 + !q2 !q1 !s

q2q1\ q0s	00	01	11	10
00	1	1	0	1
01	1	1	1	0
11	1	1	0	1
10	1	1	1	0

D0 = !q0 + q2 !q1 s + !q2 q1 s + !q2 !q1 !s + q2 q1 !s

q2q1\ q0s	00	01	11	10
00	0	0	0	0
01	0	0	1	1
11	1	1	0	0
10	1	1	1	1

Luz = q2 !q1 + !q0 q2 + !q2 q1 q0

Ejercicio 5 (se realiza en web)

Se desea mantener controladas las algas en una piscina utilizando un procesador dedicado a dicha tarea. El porcentaje de algas presente en el agua puede ser accedido a través del puerto de entrada/salida, de tamaño byte, de solo lectura en la dirección ALGAS.

Además, se dispone de un dispensador de solución anti-algas para controlar su proliferación en el agua de la piscina. El dispensador se controla mediante el bit 1 del puerto de salida DISPENSADOR de tamaño byte. Al escribir un 1 en dicho bit se abre el dispensador en caso contrario se cierra, y mientras se encuentre abierto vierte 1cc cada 100ms.

El sistema debe colocar la solución toda vez que el porcentaje supere el valor MEDIO y en caso de superar un valor MAXIMO debe aplicar un shock de solución. En caso normal se debe colocar 1 cc de solución y para el shock 4 cc. Las aplicaciones, ya sean normales o de shock, deben estar separadas por al menos 10 minutos.

Se dispone de un reloj externo que interrumpe cada 10ms invocando a la rutina timer.

Se pide:

Implemente en un lenguaje de alto nivel (preferentemente C) todas las rutinas necesarias para resolver la realidad planteada.

Solución:

```
#define estado_normal 1
#define estado_espera 2
#define estado_medio 3
#define estado_maximo 4
#define DIEZ_MINUTOS 100 * 60 * 10 // 60000
unsigned int tics;                 // para poder representar 60000 en 16 bits
char estado;

void main(){
     // instalar interrupciones
     enable();
     tics = 0;
     estado = estado_normal;
     while (true){
          if (estado == estado_normal){
               char algas = IN(ALGAS);
               if (algas > MEDIO){
                    OUT(DISPENSADOR, 2); // abro válvula
```

```
                    tics = 0;
                    if (algas > MAXIMO){
                         estado = estado_maximo;
                    }else{
                         estado = estado_medio;
                    }
               }
          }
          // opcionalmente se podría controlar que si estoy en estado medio
          // se pase a estado maximo si las algas > MAXIMO
     } // while
}
void interrupt timer(){
       tics++;
       switch(estado){
          case estado_medio:
               if (tics == 10){
                    tics = 0;
                    estado = estado_espera;
                    OUT(DISPENSADOR, 0);
               }
               break;
           case estado_maximo:
               if (tics == 40){
                    tics = 0;
                    estado = estado_espera;
                    OUT(DISPENSADOR, 0);
               }
               break;
         case estado_espera:
               if (tics == DIEZ_MINUTOS){
                    estado = estado_normal;
               }
               break;
       }
}
```

Ejercicio 6 (se realiza en web)

Se quiere utilizar una ROM para convertir el valor absoluto de un número en complemento a 2 de 16 bits en BCD empaquetado.

Especifique el tamaño y organización de la ROM necesaria para resolver el problema y escriba el programa de carga de la ROM.

Solución:

La entrada de la ROM corresponde al número en complemento a 2 de 16 bits, cuyo rango va de -32768 a 32767.
Como lo que se debe convertir es el valor absoluto la salida serán 20 bits correspondientes a los 5 dígitos BCD que se requieren para representar un número entre 0 y 32768.

Por lo tanto se requiere una ROM de 64Kx20 cuyo tamaño es de 1280Kb (es decir 160KB).

```
unsigned int ROM[65536];  // se considera int de 20+ bits, ej 32 bits

void cargarROM() {
     for(int i = -32768; i < 32767; i++) {
          unsigned int abs = i > 0 ? i : -i;
          unsigned int bcd = 0;
          for(int j = 0; j < 20; j += 4) {
               bcd |= (abs % 10) << j;
               abs /= 10;
          }
          ROM[(i&0xFFFF)] = bcd;
     }
}
```

Otra solución posible es:

```
unsigned int ROM[65536];  // se considera int de 20+ bits, ej 32 bits

void cargarROM() {
     unsigned int abs;
     unsigned int bcd;
     for(int i = 0; i < 65636; i++) {
          if (i < 32768)
               abs = i;
          else
               abs = 65536 - i;
          bcd = 0;
          for(int j = 0; j < 20; j += 4) {
               bcd |= (abs % 10) << j;
               abs /= 10;
          }
          ROM[i] = bcd;
     }
}
```