Examen de Arquitectura de Computadoras
15 de febrero de 2018

Instrucciones:

- Indique su nombre, apellido y número de cédula en todas las hojas que entregue.
- Escriba las hojas de un solo lado.
- Las hojas deben estar numeradas y en la primer hoja debe escribirse el total.
- Empiece cada ejercicio en una hoja nueva.
- No puede utilizar material ni calculadora.
- **Apague su celular.**
- La duración del examen es de tres horas. En dicho tiempo debe también completar sus datos.
  Solo se contestarán dudas de letra. No se aceptan preguntas en los últimos 30 minutos del examen.
- Para aprobar debe contestar correctamente un ejercicio entero y dos preguntas.

Pregunta 1

Describa el modelo de direccionamiento de memoria del 8086. En particular, indique como se calcula una dirección de memoria en ese modelo.

Pregunta 2

Describa las entradas y salidas típicas de una ALU. ¿Es un circuito secuencial o combinatorio? Justifique.

Pregunta3

Un código de paridad horizontal+vertical de 6x6 bits de datos, ¿puede corregir errores de 1 bit?. Justifique su respuesta.

Pregunta 4

Considere una memoria caché de 32 líneas de 64 Bytes cada una, con correspondencia asociativa por conjuntos de 2 vías y una memoria principal de 1 MByte, direccionable de a byte. Indique cuál es el hit-rate de las lecturas a memoria de las direcciones 0xC34A6, 0xC38AB y 0xC3480 (en ese orden).

Problema 1

El ***NOBODYLEFT MARKET*** desea construir un nueva sistema de cajas inteligente, en las que será el propio cliente el encargado de escanear los códigos de barras de los productos y luego realizar el pago sin la intervención de un cajero.

El sistema funciona de la siguiente manera: el cliente pasa cada producto por un escáner y a continuación lo deposita sobre una balanza donde se van acumulando los productos de la compra. El sistema espera que la balanza estabilice su medida y en ese momento verifica que el peso correspondiente al código del producto coincida con el aumento relativo en la balanza. El cliente repite este proceso para todos los artículos, y al finalizar selecciona la opción "Terminar Compra".

Se considera que la balanza está estabilizada si el peso se mantiene incambiado durante 1 segundo, en un peso **diferente** al que se medía antes de depositar un nuevo producto. Si pasado 15 segundos desde que se escaneo el producto no se estabilizó o la diferencia de peso no coincide con la esperada, el sistema desplegará un mensaje de error. En este caso el cliente deberá esperar a que un asistente

utilice una llave que destraba la máquina. Se asume que la llave se utiliza una vez que la balanza se encuentra estable, y al utilizarse el sistema queda pronto para continuar con el proceso de recepción de productos.

Se cuenta con las siguientes interrupciones:

- ***productoLeido()***: Se ejecuta cada vez que el usuario pasa un producto por el escaner.
- ***timer()***: Rutina que se ejecuta a una frecuencia de 10 Hz.

También se cuenta con los siguientes puertos:

- Puerto de 16 bits de solo lectura en la dirección CODIGO, donde se puede leer el código del último producto escaneado.
- El peso en gramos de la balanza se puede leer utilizando el puerto de 16 bits de solo lectura en la dirección PESO.
- Puerto de 8 bits de solo escritura en la dirección PANTALLA, cuyo bit menos significativo se debe utilizar para indicar un error (el bit en “1” indica error).
- Puerto de 8 bits de solo lectura en la dirección COMANDOS. El bit 0 de este byte indica que se ha presionado el botón de finalización de la compra, mientras que el bit 1 indica que se ha activado la llave que destraba en caso de error.

Se pide: implementar en un lenguaje de alto nivel, preferentemente C, todas las rutinas necesarias del sistema de cajas inteligente. Se dispone de un procesador dedicado para la tarea.

Nota: Considere que dispone de una función *pesoProducto(idProducto)*, que dado el id de un producto, devuelve su peso en gramos. Al finalizar la compra, se debe llamar a la función auxiliar *finalizarCompra().*

Problema 2

Se desea utilizar una ROM para implementar una función que determine la parte entera de un número representado en *Punto Flotante de Precisión Simple* (1 bit signo, 8 bits exponente, 23 bits parte fraccional). La parte entera a dar como salida deberá estar representada en *Complemento a 2* de 16 bits. La función también deberá indicar, mediante un bit, si la operación no puede realizarse por no ser representable la parte entera en ese formato (salida de Overflow).

Se pide:

A) Determinar tamaño y organización de la ROM necesaria para implementar la función. Construirla en base a ROMs de 1G x 8 y 1G x 4 y compuertas básicas. Se deberá utilizar la menor cantidad de ROMs posible.

B) Escribir en un lenguaje de alto nivel el programa que genera el contenido de la ROM. El lenguaje de alto nivel dispone solamente de aritmética entera y operaciones bit a bit (no dispone de aritmética de Punto Flotante).

Respuesta Pregunta 1

El modelo de memoria de 8086 es “segmentado”. Este término significa que en la memoria se identifican segmentos de 64 KBytes de tamaño asociados a registros especiales de 16 bits llamados registros de segmento, y solo se puede, en cada momento, direccionar posiciones de memoria dentro de cada segmento. Los registros de segmento son el CS (code segment), DS (data segment), SS (stack segment) y ES (extra segment).

Las direcciones físicas se forman a partir de un registro de segmento y un desplazamiento (también de 16 bits, por ser el segmento de 64 KBytes), con la siguiente operación:

dirección_física = *segmento* * 16 + *desplazamiento.*

Esto da como resultado una dirección de 20 bits.

Respuesta Pregunta 2

Las entrada típicas de una ALU son:

1. Dos entradas de operandos de N bits cada una, donde N es el tamaño de los registros.
2. Entradas de selección de operación (el número de bits depende de la cantidad de operaciones disponibles en la ALU).

Las salidas típicas de una ALU son:

1. Resultado: el resultado de la operación realizada.
2. FLAGs: las banderas calculadas en la operación (C, N, V, Z, etc).

Se trata de un circuito combinatorio, ya que la salida es resultado combinatorio de las entradas (ver por ejemplo circuito sumador, construido con compuertas básicas). Por la propia definición de la Arquitectura de Von Neumann, la ALU realiza operaciones básicas en forma combinatoria, en caso de necesitarse operaciones más complejas se realiza mediante una secuencia de operaciones de la ALU realizadas bajo el control de la Unidad de Control (que es un circuito secuencial).

Respuesta Pregunta 3

Supongamos que ocurre un error en un bit de datos al azar. Por construcción, el bit de paridad de la fila y de la columna asociadas con el bit errado darán un valor distinto del calculado originalmente, o sea se detectará como un error de paridad en la fila y columna respectiva. Luego se deduce que el bit erróneo se encuentra en la intersección de la fila y la columna indicada por los bits de paridad que difieren de su re-cálculo. Como el bit con error se eligió al azar, queda demostrado que el código corrige cualquier error de un bit.

Respuesta Pregunta 4

La memoria caché tiene 32 líneas de 64 bytes cada una, con correspondencia asociativa por conjuntos de 2 vías.

Paso 1) Determinar el largo de cada campo en una dirección dada.

BYTE: *byte/palabra individual en una línea de caché o bloque de memoria*. Las líneas son de 64 Bytes, se necesitan 6 bits para indicar la palabra. Por tanto BYTE requiere 6 bits.
SET: *conjunto de líneas asociado con un bloque*. La memoria caché tiene tiene 32 líneas y 2 vías, por tanto tiene 16 líneas/vía. Por lo cual SET requiere 4 bits.
TAG: *identificador de bloque*. La memoria principal es de 1MB y cada bloque tiene 16 líneas/vía x 64B/línea = 1024 B/vía, por lo cual entran 1MB/1024B = 2^{20} B / 2^{10} B = 2^{10} bloques. TAG requiere 10 bits.

TAG es de 10 bits, SET es de 4 bits y BYTE es de 6 bits.

TAG	LINEA/SET	BYTE
19 10	9 6	5 0

Paso 2) Hallar los campos correspondientes a cada dirección leída, determinar si los datos están en cache y los datos que se cargan.

a) C34A6 es 1100 0011 0100 1010 0110; los campos corresponden a TAG = 1100001101 (30D), LINEA = 0010 (2), BYTE = 100110 (26).
No está en caché (miss). Se carga {2:30D}.

a) C38AB es 1100 0011 1000 1010 1011; los campos corresponden a TAG = 1100001110 (30E), LINEA = 0010 (2), BYTE = 101011 (27).
No está en caché (miss). Se carga {2:30E}, la caché contiene {2:30D, 2:30E}.

a) C3480 es 1100 0011 0100 1000 0000; los campos corresponden a TAG = 1100001101 (30D), LINEA = 0010 (2), BYTE = 000000 (0).
Está en caché (hit). La caché contiene {2:30D, 2:30E}.

Hit rate: ⅓.

Solución Problema 1:

```
#define ESTADO_ESPERANDO_PRODUCTO_O_FIN_COMPRA 0
#define ESTADO_ESPERANDO_ESTABILIZAR_BALANZA 1
#define ESTADO_ESPERO_LLAVE 2
#define SEGUNDO 10
#define QUINCE_SEGUNDOS 150

char estado = ESTADO_ESPERANDO_PRODUCTO_O_FIN_COMPRA;
int tiempoEstabilizacion = 0; // cuenta los 15 segundos antes de dar error
int tiempoMismoPeso = 0; // cuenta el segundo de un mismo peso
int pesoTotalCompra = 0;

void main(){
	while (1){
		switch(estado){
			case ESTADO_ESPERANDO_PRODUCTO_O_FIN_COMPRA:
				char comandos = IN(COMANDOS);
				if (comandos & 1){
					// se presionó el botón de finalización de compra
					finalizarCompra();
					pesoTotalCompra = 0;
				}
				break;
			case ESTADO_ESPERANDO_ESTABILIZAR_BALANZA:
				// verifico que no haya variado el peso. La balanza se
				// considera estable si no varía el peso en dos segundos.
				int pesoTemp = IN(PESO);
				if (pesoTemp != pesoActual){
					tiempoMismoPeso = 0;
					pesoActual = pesoTemp;
				}
				break;
			case ESTADO_ESPERO_LLAVE:
				char comandos = IN(COMANDOS);
				if (comandos & 2){
					estado = ESTADO_ESPERANDO_PRODUCTO_O_FIN_COMPRA;
					OUT(PANTALLA, 0); // quito el error de pantalla
				}
				break;
		}
	}
}

void interrupt timer(){
	if (estado == ESTADO_ESPERANDO_ESTABILIZAR_BALANZA){
		tiempoEstabilizacion++;
		tiempoMismoPeso++;
```

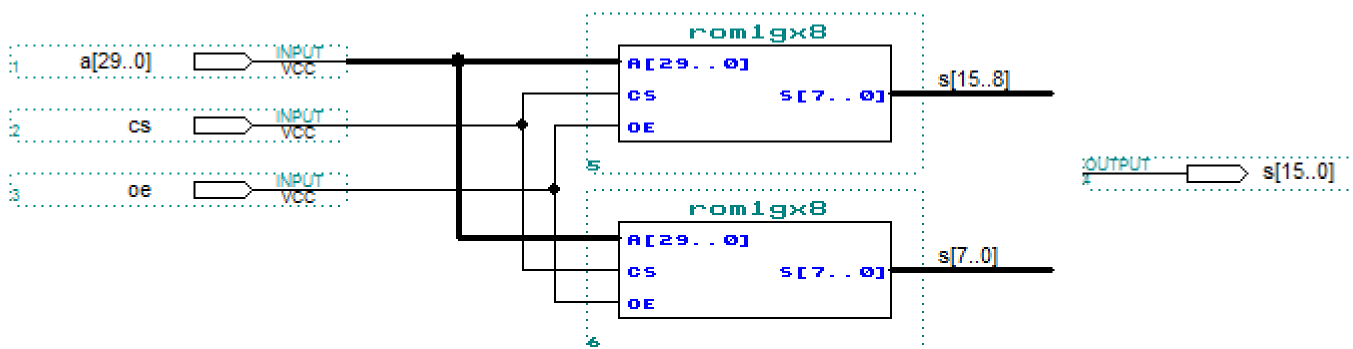
```
            // las dos maneras de salir de este estado es: o por estar 15 segundos, o
            // por estar 1 segundo con un peso estable, distinto al pesoTotalCompra
            if (tiempoMismoPeso >= SEGUNDO && pesoActual != pesoTotalCompra){
                  // pesoProducto es la función auxiliar definida en letra
                  int pesoEsperado = pesoProducto(codigoProducto);
                  if (pesoEsperado == pesoActual – pesoTotalCompra){
                        // recepciono la compra aumentando al peso total
                        pesoTotalCompra = pesoActual;
                        estado = ESTADO_ESPERANDO_PRODUCTO_O_FIN_COMPRA;
                  }else{
                        transicionEstadoError();
                  }
            }else if (tiempoEstabilizacion >= QUINCE_SEGUNDOS){
                  transicionEstadoError();
            }
      }
}

void transicionEstadoError(){
      estado = ESTADO_ESPERO_LLAVE;
      OUT(PANTALLA, 1);
}

void interrupt productoLeido(){
      // se recibió un producto, se inicializan las variables de tiempo y
      if (estado == ESTADO_ESPERANDO_PRODUCTO_O_FIN_COMPRA){
            codigoProducto = IN(CODIGO);
            estado = ESTADO_ESPERANDO_ESTABILIZAR_BALANZA;
            tiempoEstabilizacion = 0;
            tiempoMismoPeso = 0
            pesoActual = pesoTotalCompra;
      } // recibir producto en otro caso tiene comportamiento indefinido en la letra
}
```

Solución Problema 2:

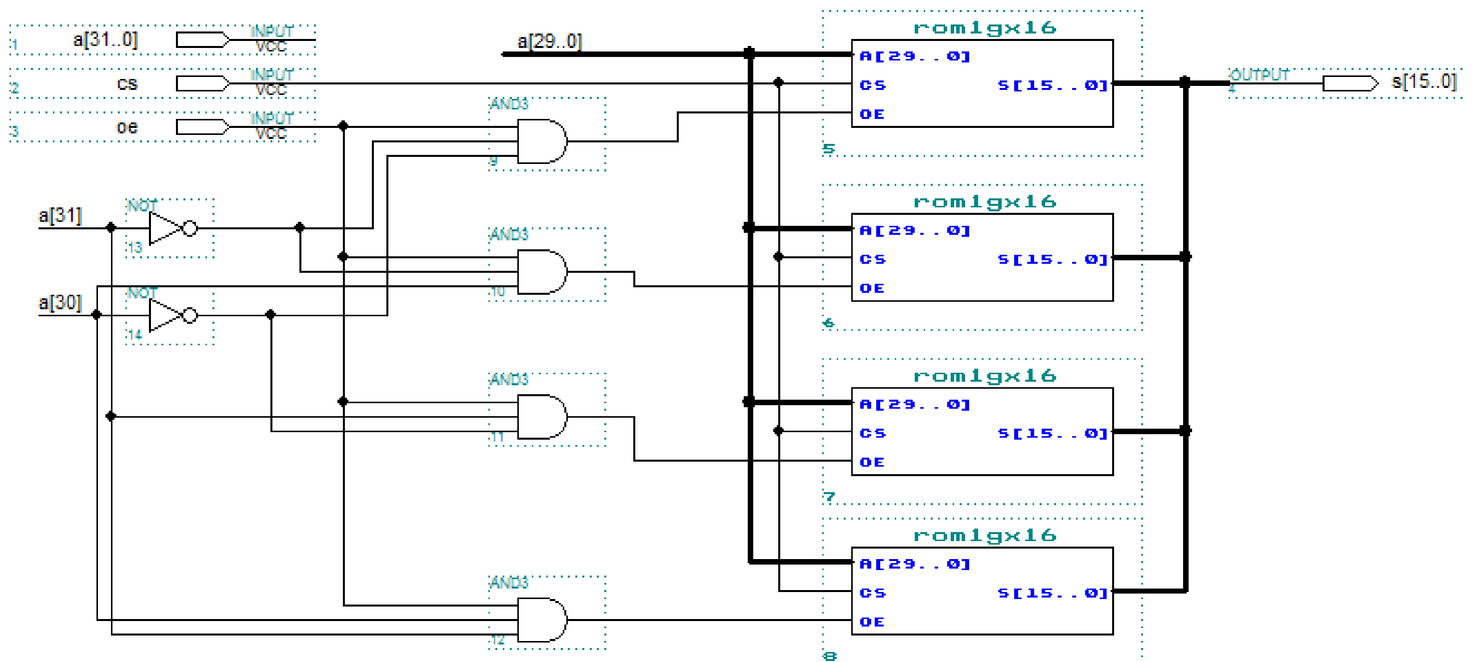
a) Necesitamos una ROM con 32 bits de entrada (E31..E0) que representan el número en punto flotante a convertir y 17 bits de salida (S16 que indica si hay Overflow y S15..S0 para representar la parte entera del número).

Por lo tanto se precisa construir una ROM de 4Gx17, que tiene un tamaño de 2^32*(2^4+1) bits = (8Gbytes + 0,5Gbytes) = 8,5GBytes

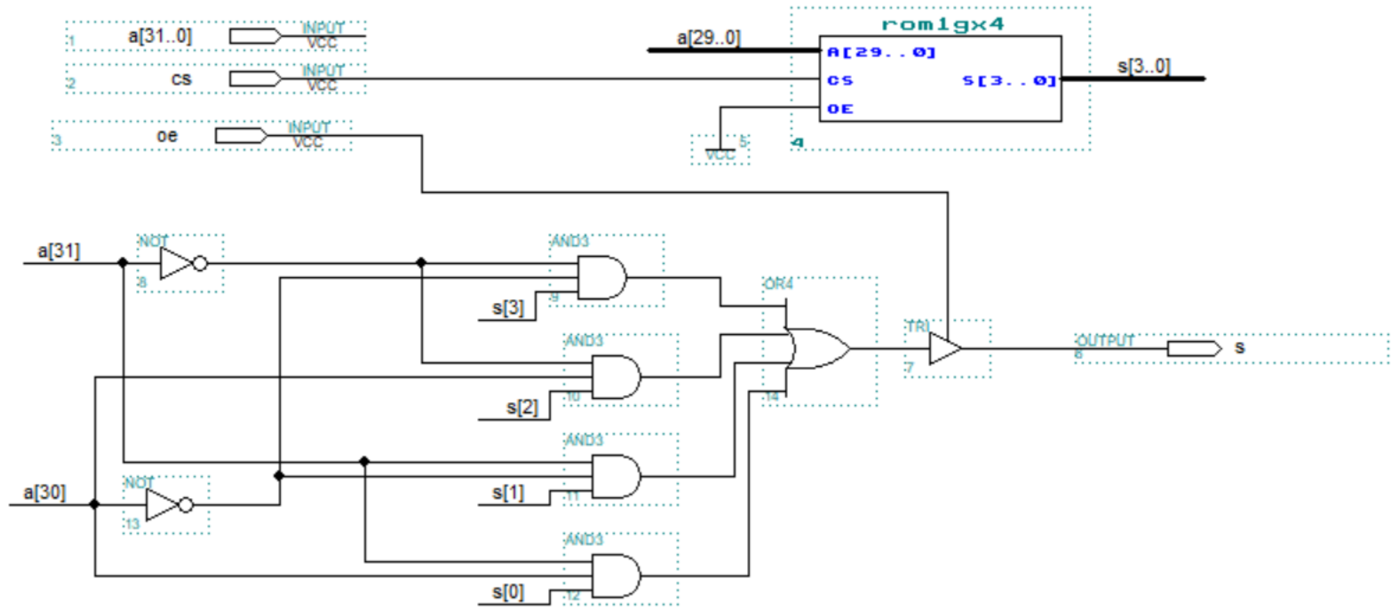
Primero construimos una ROM de 1Gx16 utilizando dos ROMs de 1Gx8, la cual llamaremos ROM A:



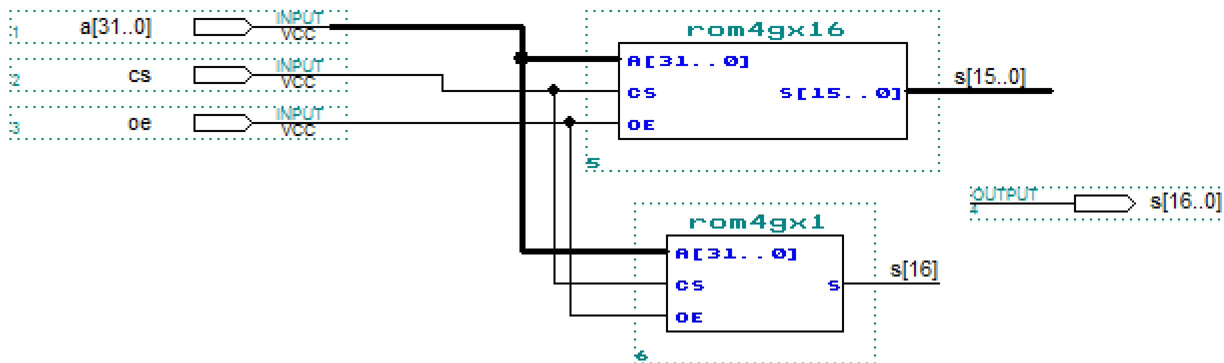
Luego construimos una ROM de 4Gx16 con 4 ROM A, la cual llamaremos ROM B:



Utilizamos una ROM de 1Gx4 para construir una de 4Gx1. La cual llamaremos ROM C:



Por último, utilizamos una ROM B y una ROM C para construir la ROM pedida:



b)

formato: [S|EXP8|FRAC23]

Los números en punto flotante de precisión simple normalizados se representan como (-1)^s*1,frac * 2^(exp8-127)

Por su parte, el rango de números en complemento a 2 de 16 bits va de -2^15 a 2^15-1

```
int ROM[1<<32]; // ints de 32 bits

void cargaROM()
{
  int entera;
  for(int codExp = 0; codExp < 255; codExp++)
  {
    int exp = codExp-127;
    for(int frac = (1<<23); frac >= 0; frac--)
    {
     entera = 0;
     overflow = 0;
     if(exp >= 0)
     {
       if(exp >= 15)
          overflow = 1<<16;
       else
          entera = 1<<exp | frac>>(23-exp);
     }
     ROM[codExp<<23|frac] = entera + overflow;
     if(exp == 15 && frac < (1<<8)) // Caso de borde
          overflow = 0;
     ROM[1<<31|codExp<<23|frac] = (-entera + overflow) & 0x1FFFF;
    }
  }
}
```

Solución alternativa:

```
int ROM[1<<32]; // ints de 32 bits

void cargaROM()
{
  for(int dir = 0; dir < 1<<32; dir++)
  {
     int signo = dir / 1<<31;
     int exp = ((dir / 1<<23) & 0xFF) - 127;
     int fracc =  dir & 0x7FFFFF;
     int overflow = 0;
     int entera = 0;
```

```
    if(exp >= 0)
    {
      if(exp >= 15)
         overflow = 1<<16;
      else
         entera = 1<<exp | frac>>(23-exp);
    }
    if((signo == 1) && (exp == 15) && (frac < (1<<8))) // Caso de borde
         overflow = 0;
    if (signo == 1) ROM[dir] = entera + overflow;
    else ROM[dir] = (-entera + overflow) & 0x1FFFF;
  }
}
```