Examen de Arquitectura de Computadoras

24 de julio del 2015

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

Defina CPU y describa los elementos básicos que la componen en la arquitectura de Von Neumann. ¿Es un circuito secuencial o combinatorio? Justifique.

Pregunta 2

Clasifique las siguientes representaciones IEEE 754 de media precisión en normalizado, desnormalizado, infinito, cero o NaN: 0xFC00, 0x4140, 0x8080 y 0x8480.

Pregunta 3

Nombre y describa los algoritmos de remplazo de bloques en memoria caché vistos en el curso. Indique para qué tipos de correspondencia aplican.

Pregunta 4

Defina el formato de instrucción y los códigos de operación para la siguiente arquitectura RISC de 16 bits con 8 registros.

Instrucción	Resultado	Instrucción	Resultado	Instrucción	Resultado
AND RegX, RegY, RegZ	RegZ = RegX AND RegY	ADD RegX, RegY, RegZ	RegZ = RegX + RegY	LOAD RegX, RegY	RegY = Mem[RegX]
OR RegX, RegY, RegZ	RegZ = RegX OR RegY	SUB RegX, RegY, RegZ	RegZ = RegX - RegY	STORE RegX, RegY	Mem[RegY] = RegX
NOT RegX, RegY	RegY = NOT RegX	DIV RegX, RegY, RegZ	RegZ = RegX DIV RegY	MOV RegX, RegY	RegY = RegX
XOR RegX, RegY, RegZ	RegZ = RegX XOR RegY	MOD RegX, RegY, RegZ	RegZ = RegX MOD RegY	MOVI Inm8, RegX	RegX = Inm8(8 Bits)

Problema 1

La cadena de supermercados **Hipermercado Petit** desea diseñar su propio sistema de guardabultos electrónico, basado en un microcontrolador dedicado.

El guardabultos será utilizado por los clientes para guardar sus pertenencias antes de entrar al supermercado. La diferencia con los guardabultos tradicionales es que el sistema de apertura del mismo será electrónico.

El funcionamiento del guardabultos es el siguiente:

Para obtener un casillero, el cliente debe presionar un botón. Si hay un casillero disponible, el sistema abre su puerta para que el cliente guarde sus pertenencias y espera a que la cierre. Si no hay un casillero libre enciende un LED durante 5 segundos para indicar la indisponibilidad.

Cuando el cliente cierra la puerta del casillero, el sistema imprime un ticket con un número (generado en forma aleatoria) representado en código de barras, el que será utilizado luego para abrir el casillero.

Cuando el cliente desea retirar sus pertenencias, presenta el ticket a un lector de código de

barras para que el sistema pueda identificar el casillero correspondiente y proceder a abrir la puerta.

Si el cliente no cierra el casillero en menos de 45 segundos, ya sea al momento de colocar sus pertenencias o retirarlas, se activa una alarma. La alarma permanecerá activada hasta que se cierre la puerta que la originó.

Si se vuelve a presionar el botón mientras se está procesando un pedido anterior, o mientras se está en condición de alarma, el sistema ignora el nuevo pedido.

El sistema cuenta con los siguientes elementos:

- El botón de apertura, que al presionarse genera una interrupción que será atendida por la rutina **`boton()`**.
- 16 (dieciséis) casilleros, cada uno con un sensor que permite saber si está abierto o cerrado. Estos sensores están disponibles en los respectivos bits del puerto de E/S de solo lectura de 16 bits en la dirección **ESTADO_CASILLEROS** (1 indica abierto, 0 cerrado). Además, cada casillero tiene un sistema de apertura electromecánico, el cual es accesible en los bits del puerto de E/S de solo escritura de 16 bits en la dirección **APERTURA_CASILLEROS** (escribiendo un 1 en un bit de dicho puerto abre el casillero correspondiente).
- Un dispositivo para impresión y lectura de tickets. Para imprimir el ticket se debe escribir un número de 16 bits en el puerto de E/S de lectura/escritura en la dirección **TICKET**. Al escribir dicho puerto, el sistema de impresión se encarga de imprimir el ticket con dicho código numérico. Luego, cuando el cliente presenta el ticket al lector, el dispositivo deja en el puerto **TICKET** el código numérico leído y genera una interrupción que es atendida por la rutina **`ticket()`**.
- Un LED y una señal sonora (alarma), utilizados para informar del estado del sistema al cliente. El pitido se controla con el bit 0 del byte de E/S de solo escritura en la dirección **PANEL,** mientras que el LED se controla con el bit 1 del mismo puerto.
- Un timer que interrumpe con frecuencia de 50 hz invocando a la rutina **`tiempo()`**.

Se pide:

Escribir en un lenguaje de alto nivel (preferentemente C) todas las rutinas necesarias para implementar el sistema descripto.

Se dispone de la siguiente rutina auxiliar:

- **`unsigned short random();`** que devuelve un nuevo número aleatorio de 16 bits para ser usado como código numérico del ticket. Se supone que esta rutina no repite números en invocaciones consecutivas hasta no haber devuelto todos los números posibles.

Problema 2

Se pretende almacenar en una memoria ROM la parte entera del valor medio de dos números enteros representados en binario de 8 bits.

Se pide:

Parte A) Especifiar tamaño de la ROM, entradas/salidas y construirla disponiendo de compuertas básicas y de las siguientes memorias

1 ROM 16K x 8
2 ROMs 16K x 4
1 ROM 8K x 16
2 ROMs 32K x 2

Implementar en alto nivel el código de carga de la ROM

Parte B) Compilar el código de carga de la parte anterior en Intel 8086. Asuma que la ROM comienza en la posición 0 del ES.

Solución

Pregunta 1

La CPU es la unidad central de procesamiento, la encargada de ejecutar las instrucciones en una máquina de Von Neumann.
La misma contiene:

- ALU o unidad aritmético-lógica, circuito combinatorio encargado de realizar diferentes operaciones.
- Banco de Registros, elementos de memoria de rápido acceso.
- Unidad de Control, circuito secuencial encargado de llevar adelante el ciclo de instrucción.

La CPU es un circuito SECUENCIAL dado que sus salidas no solo dependen de las entradas actuales, sino también de las entradas anteriores. Un ejemplo claro de esto, es que lo que una computadora hace (salidas de la CPU) depende de varias instrucciones leídas de memoria (entradas anteriores a la CPU).

Pregunta 2

Número	S	E	M	TIPO
0xFC00	1	11111	0000000000	Menos infinito
0x4140	0	10000	0101000000	Normalizado positivo
0x8080	1	00000	0010000000	Desnormalizado negativo
0x8480	1	00001	0010000000	Normalizado negativo

Pregunta 3

Menos Recientemente Usado (LRU = Least Recently Used)
Este algoritmo selecciona para reemplazar, dentro de las líneas posibles, la que tenga el bloque que haya sido accedido hace más tiempo. Una implementación sencilla de este algoritmo para una memoria cache asociativa por conjuntos de 2 vías es usar un bit que indica cual fue la última línea que se accedió. En cada acceso se actualizan los bits de ambas líneas del conjunto en forma apropiada.

FIFO (First In First Out)
En este caso se selecciona la línea que contiene el bloque que haya sido traído primero desde la memoria principal (el más antiguo), sin importar si fue accedido ni que tantas veces lo fue. Una manera de implementar este algoritmo en hardware es mediante un buffer circular (con un puntero de circular por conjunto que señala la línea a reemplazar que se actualiza en cada reemplazo).

Menos Frecuentemente Utilizado (LFU = Least Frequently Used)
Este algoritmo utiliza la cantidad de veces que han sido accedidos los bloques de las líneas candidatas a ser reemplazadas. Para su implementación se pueden utilizar contadores en cada línea.

Random
El algoritmo random selecciona el bloque a reemplazar mediante una técnica aleatoria (normalmente pseudo-aleatoria por razones de implementación).

Estas técnicas se utilizan en aquellas correspondencias donde existe más de una alternativa posible para alojar un bloque de memoria en la Cache: totalmente asociativa o asociativa por conjuntos de n-vías.

Pregunta 4

Son 12 operaciones, por lo que se precisan 4 bits de código de operación.
Son 8 registros, por lo que para identificar un operando tipo registro precisamos 3 bits.
Es una arquitectura RISC por lo que todas las instrucciones son del mismo largo (16 bits).
Formato de instrucción, hay tres tipos únicamente:
3 operandos registro(AND, OR, XOR, ADD, SUB, DIV, MOD):
[OPCODE 4 BITS]000[REGX 3BITS][REGY 3BITS][REGZ 3BITS]
2 operandos registro(NOT, LOAD, STORE, MOV):
[OPCODE 4 BITS]000[REGX 3BITS][REGY 3BITS]000
1 operando registro y otro inmediato (MOVI)
[OPCODE 4 BITS]0[REGX 3BITS][INM8 8BITS]
OPCODES:
AND 0000, OR 0001, NOT 0010, XOR 0011, ADD 0100, SUB 0101, DIV 0110, MOD 0111, LOAD 1000, STORE 1001, MOV 1010, MOVI 1011

Problema 1

```
# define ESTADO_ESPERO_BOTON_TICKET 0
# define ESTADO_ESPERO_CIERRE_CASILLERO_BOTON 1
# define ESTADO_ERROR_SIN_CASILLEROS 2
# define ESTADO_ESPERO_CIERRE_CASILLERO_TICKET 3

# define CUARENTA_Y_CINCO_SEGUNDOS 2250
# define CINCO_SEGUNDOS 250

# define APERTURA_CASILLEROS ...
# define ESTADO_CASILLEROS ...
# define PANEL ...
# define TICKET ...

# define MASK_PITIDO 1
# define MASK_LED 2

int estado;
int hayBoton;
int casilleroAbriendo;
int enAlarma;
unsigned short codigoTicket;

struct casillero{
  char libre;
  unsigned short codigoApertura;
}

casillero[16] casilleros;

void main(){

  estado = 0;
  hayBoton = 0;
  hayTicket = 0;
  enAlarma = 0;
  for (int j = 0; j < 16; j++){
       casilleros.libre = 1;
  }

  // instalo interrupciones
  enable();

  while(1){

       switch(estado){
            case ESTADO_ESPERO_BOTON_TICKET:
                 if (hayBoton){
                      hayBoton = 0;
                      // presionaron el boton, me fijo si hay locker libre
                      int j;
                      for (j = 0; j < 16; j++){
                           if (casilleros[j].libre) break;
                      }
```

```
                if (j == 16){ // No hay casilleros libres, doy error
                    OUT(PANEL, MASK_LED);
                    estado = ESTADO_ERROR_SIN_CASILLEROS;
                    contando = 0;
                    tics = 0;
                }else{ // El casillero j está libre, lo uso
                    casilleroAbriendo = j;
                    casilleros[j].libre = 0;
                    OUT(APERTURA_CASILLEROS, 1 << j); // abro el casillero
                    contando = true;
                    tics = 0;
                    estado = ESTADO_ESPERO_CIERRE_CASILLERO_BOTON;
                }
            }else if (hayTicket){
                hayTicket = 0;
                // Busco el codigo correspondiente para saber cuál abrir
                int j;
                for (j = 0; j < 16; j++){
                    if ((casilleros[j].codigoApertura == codigoTicket)
                        && casilleros[j].libre == 0){
                        casilleroAbriendo = j;
                        break;
                    }
                }
                // Abro el casillero y lo marco como libre
                casilleros[j].libre = 1;
                OUT(APERTURA_CASILLEROS, 1 << j); // abro el casillero
                contando = true;
                tics = 0;
                estado = ESTADO_ESPERO_CIERRE_CASILLERO_TICKET;
            }
            break;
        case ESTADO_ESPERO_CIERRE_CASILLERO_BOTON:
        case ESTADO_ESPERO_CIERRE_CASILLERO_TICKET:
            short estado = IN(ESTADO_CASILLEROS);
            if (!(estado & (1 << casilleroAbriendo))){
                estado = ESTADO_ESPERO_BOTON_TICKET;
                contado = false;
                if (enAlarma){ // Si activé la alarma, la apago
                    OUT(PANEL, 0);
                    enAlarma = 0;
                }
                if (estado == ESTADO_ESPERO_CIERRE_CASILLERO_BOTON){
                    // Si el casillero se abrió por botón imprimo el
                    // ticket
                    unsinged short rand = random();
                    OUT(TICKET, rand);
                    casilleros[casilleroAbriendo].codigoApertura = rand;
                }
            }else if (tics > CUARENTA_Y_CINCO_SEGUNDOS){
                OUT(PANEL, MASK_PITIDO | MASK_LED);
                contando = 0;
                tics = 0;
                enAlarma = 1;
            }
            break;
        case ESTADO_ERROR_SIN_CASILLEROS:
```

```
                        if (tics > CINCO_SEGUNDOS){
                              OUT(PANEL, 0);
                              contando = 0;
                              tics = 0;
                              estado = ESTADO_ESPERO_BOTON_TICKET;
                        }
                        break;
                  }

            }

      }

      void interrupt timer(){
            if (contando){
                  tics++;
            }
      }

      void interrupt boton(){
            hayBoton = 1;
      }

      void interrupt ticket(){
            hayTicket = 1;
            codigoTicket = IN(TICKET);
      }
```

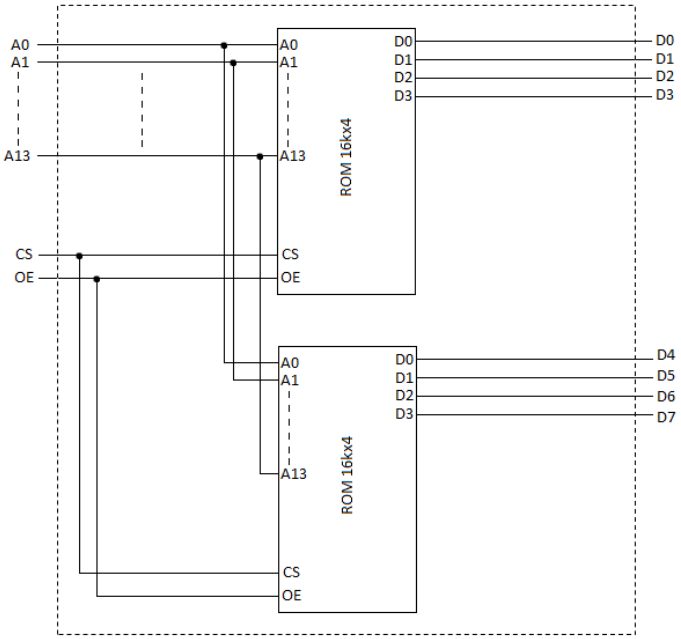
Problema 2

Parte a)

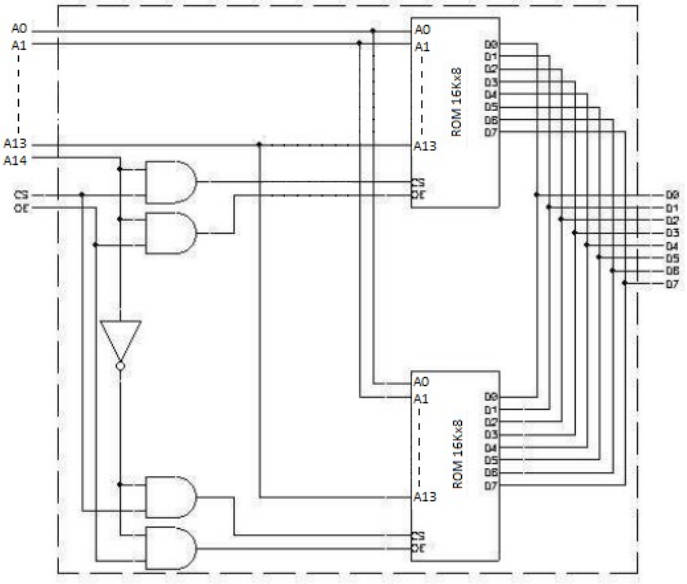
La ROM tendrá 16 bits de entrada de datos, dos entradas de control (CS y OE) y 8 bits de salida. Lo cual tiene como resultado una memoria de 64Kx8.

A continuación se describe la construcción de la memoria.

Paso 1: A partir de ambas memorias de 16Kx4 construimos una memoria de 16Kx8.



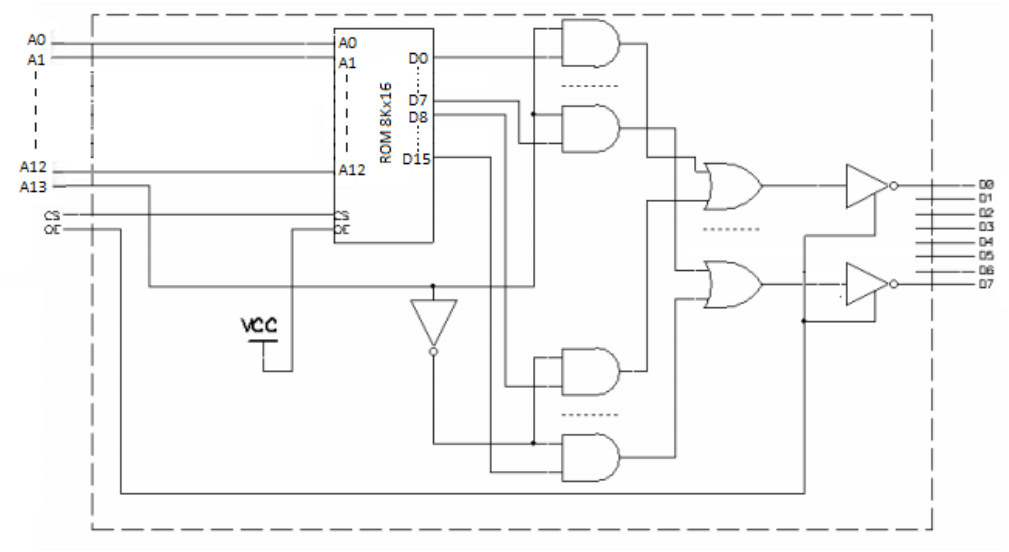
Paso 2: Luego utilizamos lasmemorias de 16Kx8 (la disponible y la construida en el Paso 1) para construir una memoria de 32Kx8.



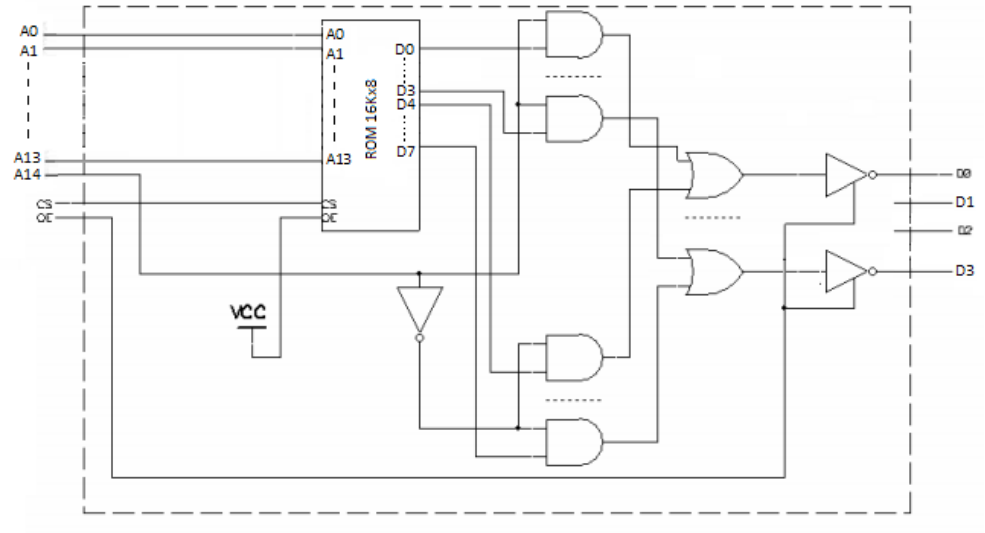
Teniendo hasta el momento las siguientes memorias:

- 1 ROM 32Kx8 (construida en los pasos anteriores)
- 1 ROM 8K x 16
- 2 ROMs 32K x 2

Paso 3: Utilizando la memoria de 8Kx16 construimos una memoria de 16Kx8.



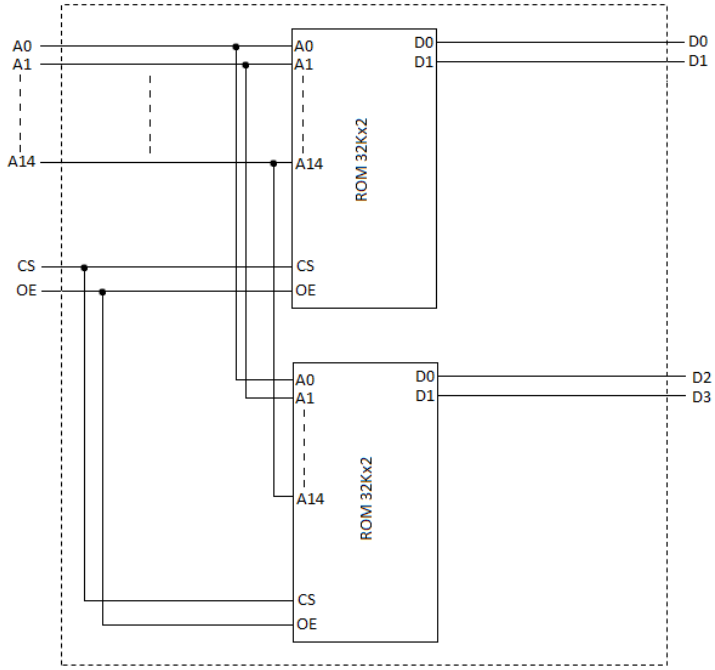
Paso 4: Con la memoria de 16Kx8 del paso anterior (paso 3) construimos una memoria de 32Kx4.



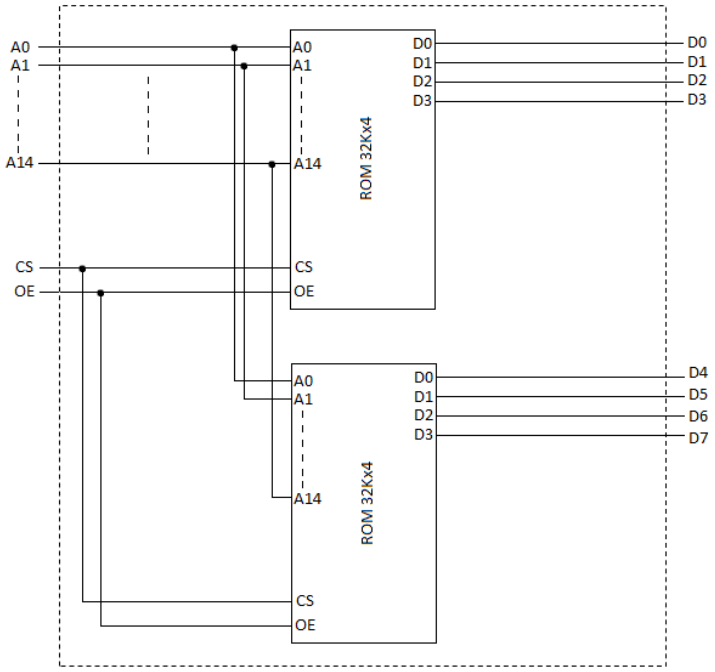
Teniendo hasta el momento las siguientes memorias:

- 1 ROM 32Kx8 (construida en los pasos 1 y 2)
- 1 ROM 32Kx4 (construida en los pasos 3 y 4)
- 2 ROMs 32K x 2

Paso 5: Con ambas memorias de 32Kx2 construimos una memoria de 32Kx4.



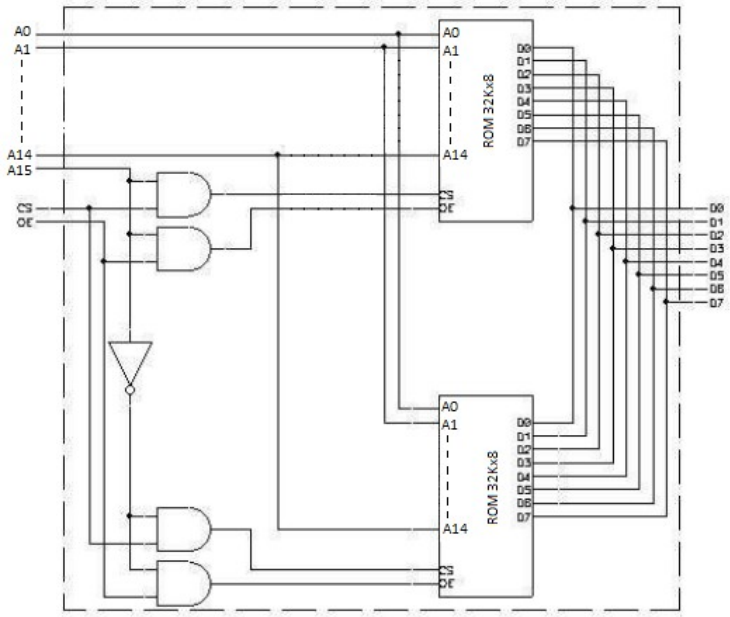
Paso 6: Con las memorias de 32Kx4 construidas en los pasos 4 y 5 construimos una memoria de 32Kx8.



Teniendo hasta el momento las siguientes memorias:

- 1 ROM 32Kx8 (construida en los pasos 1 y 2)
- 1 ROM 32Kx8 (construida en los pasos 3, 4, 5 y 6)

Paso 7: Por último utilizando las memorias de 32Kx8 de los pasos 2 y 6 construimos la memoria resultante de 64Kx8.



Nota: los pasos 3 y 4 se pueden hacer en uno solo, utilizando compuertas AND de 3 entradas y decodificando dos bits de dirección a la vez.

Código de carga en alto nivel:

```
void cargarROM(){

  unsigned short  posMem, res;
  unsigned char memROM[65536];

  for (short i = 0; i < 256; i++) {

    for (short j = 0; j < 256; j++) {

      res = (i+j)/2;
      posMem = i*(1 << 8) + j;
      memROM[posMem] = res;

    }

  }

}
```

Parte b)

```
proc cargarROM
  push AX          ; salvo contexto (aunque no es estrictamente necesario – la letra no lo pide)
  push BX
  push CX
  push DX


  MOV AX, 0        ; AX = i = 0
firstLoop:

  MOV CX, 0        ; CX = j = 0
secondLoop:

  mov DX, AX
  add DX, CX       ; DX = i + j
  shr DX, 1        ; DX = DX/2
  mov BX, AX       ; BX = i
  push CX          ; salvo CX temporalmente
  mov CL, 8
  shl BX, CL       ; i * (1 << 8)
  pop CX           ; restauro CX
  add BX, CX       ; BX = posMem = i * (1 << 8) + j
  mov ES:[BX], DL  ; guardo valor en memoria
  inc CX
  cmp CX, 256      ; si j no alcanzó 256 sigo en el segundo for
  jnz secondLoop
  inc AX
  cmp AX, 256      ; si i no alcanzó 256 sigo en el segundo for
  jnz firstLoop
  pop DX           ; restauro contexto
  pop CX
  pop BX
  pop AX           ; finalizo procedimiento
endp
```