Examen de Arquitectura de Computadoras

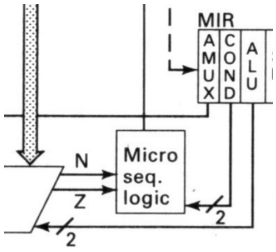
19 de febrero del 2014

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

Determine la función y el circuito para el componente "*Micro Seq.* Logic" de la organización de la CPU MIC-1. Asuma que MMUX selecciona el valor de *Increment* si su entrada de control vale 0. **No es necesario minimizar.**



Pregunta 2

En una arquitectura 8086 se tiene la siguiente definición:

```
typedef struct {
  char ci[8];
  short edad;
  char sexo;
} Persona;
Persona agenda[5];
```

Indique cómo queda en memoria la variable agenda considerando que se encuentra almacenada a partir del segmento 0x1230 y desplazamiento 0x0000.

Pregunta 3

Describa las funciones de un Multiplexor, un Demultiplexor y de un Decodificador. Realice los dibujos de los siguientes circuitos:

- Demultiplexor de 1 entrada de control.
- Decodificador de dos entradas.

Pregunta 4

¿Cuáles son las tres partes fundamentales de una CPU vistas en el curso?. ¿Qué función tiene cada una?.

Problema 1

Se considera un árbol binario cuyo nodo es definido de la siguiente manera:

```
struct nodo {
      byte indIzquierdo;
      byte indDerecho;
      short numero;
};
```

donde:

`indIzquierdo`, `indDerecho` son índices de 8 bits a un array de nodos. El primero indica el hijo izquierdo y el segundo el hijo derecho.

`numero` es un entero de 16 bits

El árbol no tiene porque estar balanceado. El valor 0 en cualquiera de los índices (derecho o izquierdo) significa que el nodo no tiene un sucesor por la correspondiente rama del árbol (no existe el nodo 0).

Se pide:

Parte a) - Escribir en un lenguaje de alto nivel una función recursiva que calcula la profundidad del árbol (largo máximo de caminos entre la raíz y un nodo). La función recibe como argumento el índice al (sub)árbol a recorrer y el puntero a memoria a partir del cual está almacenado el array. Compilar la rutina en assembler 8086. El programa llamador hace la siguiente invocación:

```
PUSH segmento arbol
PUSH offset arbol
PUSH indice
CALL busco_profundidad
POP profundidad
```

el resultado se devuelve en el stack y los argumentos deben retirarse del stack.

Parte b) - Calcular el tamaño mínimo de stack para que la función pueda ser ejecutada en todos los casos, cualquiera sea el tamaño del árbol.

Problema 2

Se desea utilizar una memoria ROM para almacenar el contenido de una función x=log(a) siendo a un entero sin signo de 32 bits y x codificado en punto flotante de precisión simple usando la norma IEEE 754.

Se pide:

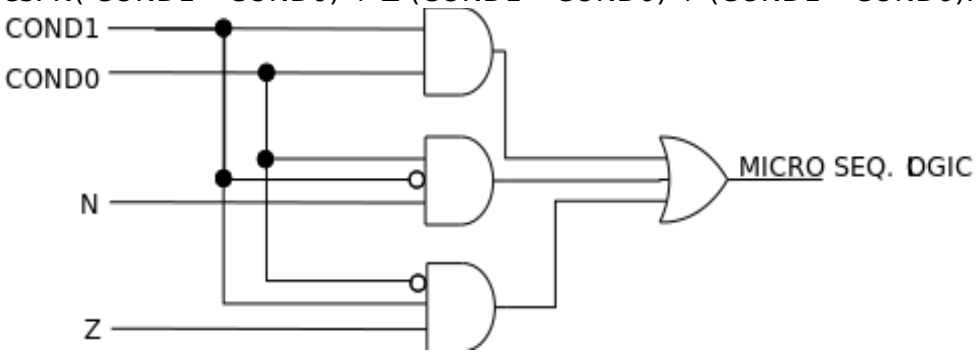
Parte a) - Determinar el tamaño y la organización de la ROM especifiando el significado de sus entradas y salidas. Construya la memoria requerida en base a ROMs de 1Gx16.

Parte b) - Interesa que los accesos a la memoria construida sean balanceados entre las memorias que la componen. El balanceo debe ser tal que en 4 accesos de direcciones consecutivas respondan al menos una única vez cada una de las memorias componentes. ¿Cumple su circuito estas características? Si es así, justifique. Si no, indique las modificaciones que permitirían realizarlo y justifique.

Solución

Pregunta 1

La función es: N($\overline{COND1}$ * COND0) + Z (COND1 * $\overline{COND0}$) + (COND1 * COND0).



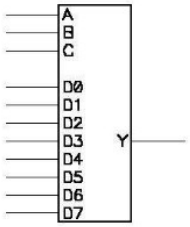
Pregunta 2

Dirección	Dato	
0x12300	CI[0]	agenda[0]
0x12301	CI[1]	
0x12302	CI[2]	
0x12303	CI[3]	
0x12304	CI[4]	
0x12305	CI[5]	
0x12306	CI[6]	
0x12307	CI[7]	
0x12308	Edad (baja)	
0x12309	Edad (alta)	
0x1230A	Sexo	
0x1230B-0x12312	CI	agenda[1]
0x12313-0x12314	Edad	
0x12315	Sexo	
0x12316-0x1231D	CI	agenda[2]
0x1231E-0x1231F	Edad	
0x12320	Sexo	
0x12321-0x12328	CI	agenda[3]
0x12329-0x1232A	Edad	
0x1232B	Sexo	
0x1232C-0x12333	CI	agenda[4]
0x12334-0x12335	Edad	
0x12336	Sexo	

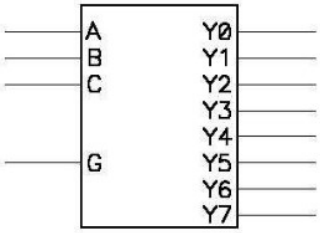
Pregunta 3

a)

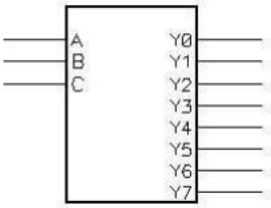
El circuito multiplexor es un circuito con N + 2^N entradas y 1 salida. Se denomina multiplexor porque es capaz de presentar en una sola variable Y cualquiera de las 2N entradas. Esto tiene aplicaciones cuando la salida Y es un "canal principal de comunicación", mientras que las entra das D son "sub-canales de comunicación". Este circuito permite que variando en el tiempo las entradas de control logremos dividir la utilización del canal principal entre los sub-canales. Esta técnica se denomina TDM (Time Division Multiplexing). De allí que el circuito se denomine multiplexor.



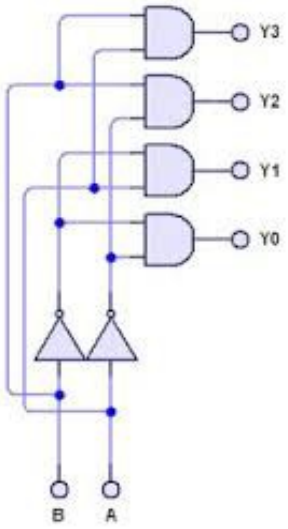
El circuito demultiplexor realiza la tarea inversa al multiplexor. Posee 1 entrada de datos, N entradas de control y 2^N salidas, según el símbolo.



El circuito decodificador es un circuito con N entradas y 2^N salidas. Para el caso de N = 3 tiene el siguiente símbolo. Dada una combinación de unos y ceros a la entrada todas las salidas estarán en 0 salvo la que corresponda al número binario coincidente con la combinación de entradas.



b)



Pregunta 4

Los tres componentes fundamentales de la CPU vistos en el curso son la ALU, la Unidad de Control y el banco de registros.

La ALU es un circuito combinatorio, que implementa operaciones aritméticas (suma, resta, etc) y lógicas (and, or, xor, etc).

La Unidad de Control es un circuito secuencial encargado de generar las señales que controlarán el resto de la CPU. Es el encargado de realizar el 'ciclo de instrucción'.

El banco de registros es un conjunto de registros con diferentes funcionalidades según su categoría. Algunos son programables y operan simplemente como posiciones de memoria de acceso más rápido, mientras que otros mantienen el estado de la CPU.

Problema 1

Parte a)

```
struct nodo {
      byte indIzq;
      byte indDer;
      int numero;
}

int buscoProfundidad(nodo [] arbol, byte indice) {
      int profIzq, profDer;
      if (indice == 0)
            return(0);
      else {
            profIzq  = buscoProfundidad(arbol, arbol[indice].indIzq) + 1;
            profDer = buscoProfundidad(arbol, arbol[indice].indDer) + 1;
            if (profIzq > profDer)
                  return(profIzq);
            else
                  return(profDer);
      } // else
} // buscoProfundidad

buscoProfundidad   proc
            pop    dx           ; dir de retorno
            pop    bx           ; indice
            pop    di           ; offset del arbol
            pop    es           ; segmento del arbol
            push   dx
            cmp    bx, 0
            jne    noEsHoja
            pop    dx
            push   bx           ; bx es 0 y debo retornar 0
            push   dx
            ret
noEsHoja:
            mov    si, bx
            shl    si, 1
            shl    si, 1        ; indice * 4 (cada nodo ocupa 4 bytes)
            add    si, di       ; sumo el offset del arbol
            xor    ah, ah
            mov    al, es:[si]
            push   si           ; guardo el offset del nodo actual (contexto)
            push   es           ; paso argumento (segmento del arbol)
            push   di           ; paso argumento (desplazamiento del arbol)
            push   ax           ; paso argumento (indice izquierdo)
            call   buscoProfundidad
            pop    ax           ; obtengo resultado
            inc    ax           ; calculo profIzq
            pop    si           ; recupero offset del nodo actual
            push   ax           ; guardo profIzq (contexto)
            xor    ah, ah
            mov    al, es:[si + 1]
            push   es           ; paso argumento (segmento del arbol)
            push   di           ; paso argumento (desplazamiento del arbol)
            push   ax           ; paso argumento (indice derecho)
            call   buscoProfundidad
            pop    ax
            inc    ax           ; calculo profDer
            pop    bx           ; recupero profIzq
            cmp    ax, bx
            jle    esProfIzq
            pop    dx           ; direccion de retorno
```

```
            push   ax           ; coloco resultado
            push   dx
            ret
esProfIzq:
            pop    dx           ; direccion de retorno
            push   bx           ; coloco resultado
            push   dx
            ret
buscoProfundidad    endp
```

Parte b)

Como los índices son de tipo byte, el árbol tiene a lo sumo 255 nodos (el 0 está excluido, por ser la marca de fin).

La profundidad mayor es cuando el árbol degenera en una lista. Para el código implementado es indiferente (como peor caso) que la lista sea por la izquierda o por la derecha (el consumo de stack es el mismo).

El consumo de stack es el siguiente:

- llamada recursiva: 2 palabras (dir de retorno, contexto: índice ó resultado intermedio)
- llamada final: 4 palabras (dir retorno y argumentos)

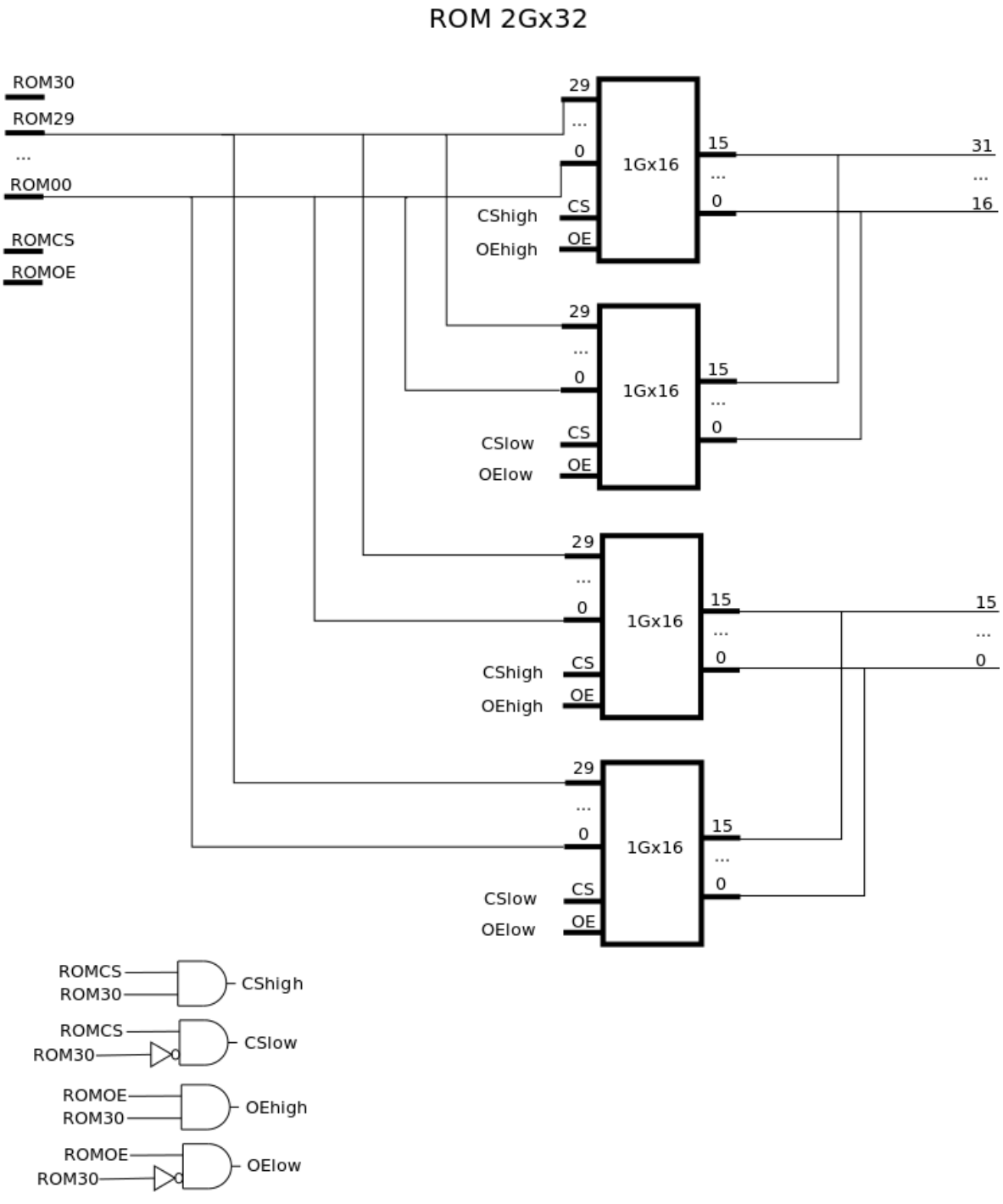
El consumo stack = n recursivas + final, para n = 255 => consumo máximo stack = 255 x 2 + 4 = 514 palabras. Entonces el requerimiento mínimo de stack para que la función pueda ser ejecutada en todos los casos es 1028 bytes.

Problema 2

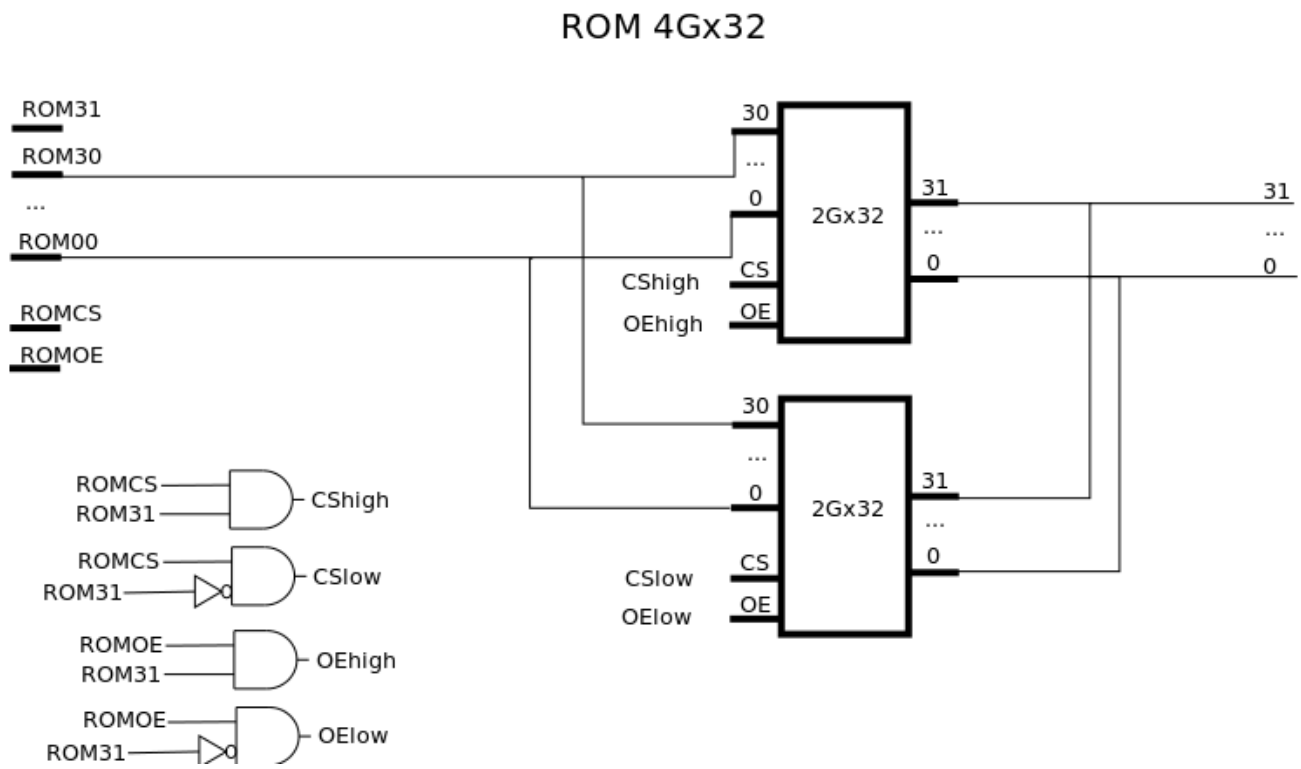
Parte a)

El circuito para implementar la memoria ROM tendrá 32 líneas de entradas de datos para los 32 bits del número entero a y tendrá 32 líneas de salida para el número en punto flotante de precisión simple x. En base a esto, podemos saber que el tamaño de la ROM resultante es de 4Gx32.

Para implementarla debemos utilizar 8 memorias de 1Gx16.


ROM 2Gx32
ROM30
ROM29
...
ROM00
ROMCS
ROMOE
29
...
0
CShigh
OEhigh
CS
OE
1Gx16
15
...
0
31
...
16
CSlow
OElow
15
...
0

ROMCS
ROM30
CShigh
CSlow
ROMOE
OEhigh
OElow

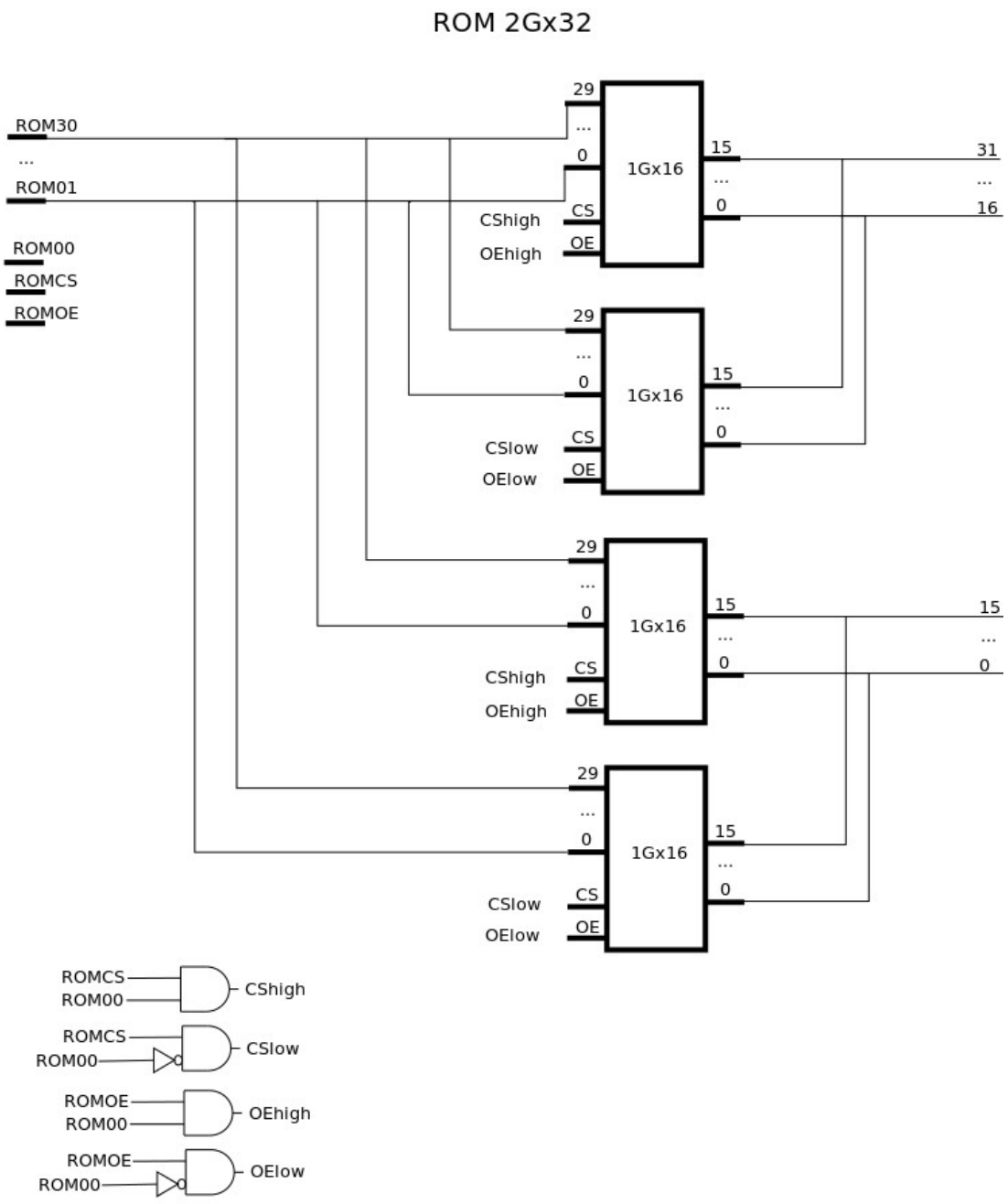


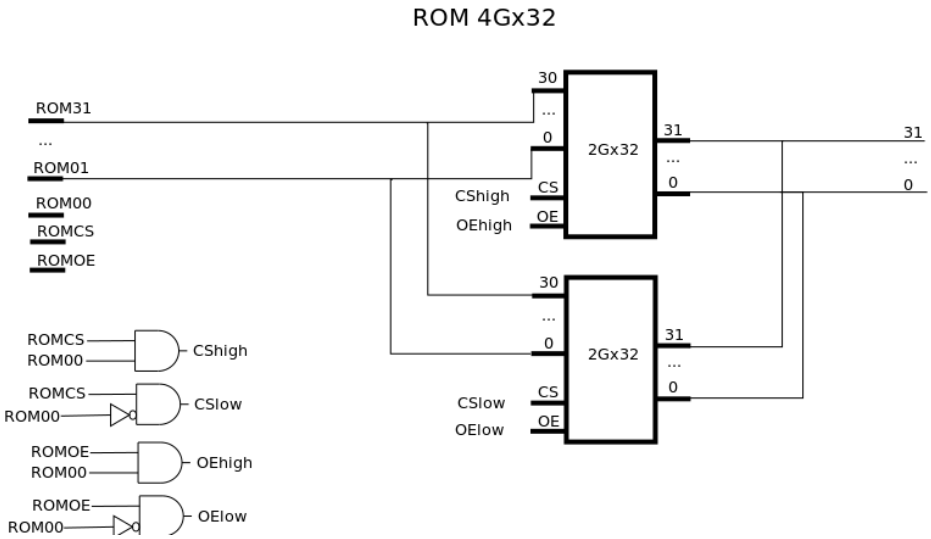
Parte b)

La memoria construida no permite el balanceo entre las diferentes memorias internas. Esto puede ver si se accede a las direcciones 0x00000000, 0x00000001, 0x00000002 y 0x00000003 que siempre utilizan la misma memoria de 2Gx32 (dado que el bit más significativo es 0).

Para corregir esto debemos utilizar los bits más significativos para las direcciones y dejar los menos significativos para la elección de las memorias ROM. De esta forma, los 2 bits menos signficativos de la dirección son los que terminan eligiendo la pareja adecuada de memorias 1Gx16.

Como en 4 direcciones consecutivas siempre se dan las 4 posibles combinaciones en los bits menos significativos, sabemos que se acceden todas las memorias internas.


ROM 2Gx32
ROM30
...
ROM01
ROM00
ROMCS
ROMOE
29
...
0
1Gx16
15
...
0
CShigh
CS
OEhigh
OE
31
...
16
CSlow
OElow
15
...
0
ROMCS
ROM00
CShigh
CSlow
ROMOE
OEhigh
OElow


ROM 4Gx32
ROM31
...
ROM01
ROM00
ROMCS
ROMOE
30
...
0
CShigh
CS
OEhigh
OE
2Gx32
31
...
0
30
...
0
CSlow
CS
OElow
OE
2Gx32
31
...
0
31
...
0
ROMCS
ROM00
CShigh
ROMCS
ROM00
CSlow
ROMOE
ROM00
OEhigh
ROMOE
ROM00
OElow