Universidad Autónoma de Baja California

Facultad de ciencias químicas e ingeniería

Materia.

Microprocesadores y Microcontroladores.

Maestro.

Garcia Lopez Jesus Adan.

Alumno

Gonzalez Cardiel Luis Enrique

Matricula:

1217258

Grupo:

561

Trabajo:

Practica No.3

28/09/2018

Práctica No. 3

• Sección de Memoria (Prueba de memoria RAM)

Objetivo: El alumno hará uso de una técnica de prueba de memoria aplicándolo en un programa de prueba de memoria RAM.

Material: - Memoria RAM y Latch para T-Juino.

Equipo:
- Computadora Personal
- Tarjeta T-Juino.
- Protoboard
- Una Memoria RAM (2K u 8K)

Teoría: * * * algoritmos de prueba para memoria RAM * * *
* * * funciones peek y poke() en arquitectura x86 (16bits) * * *

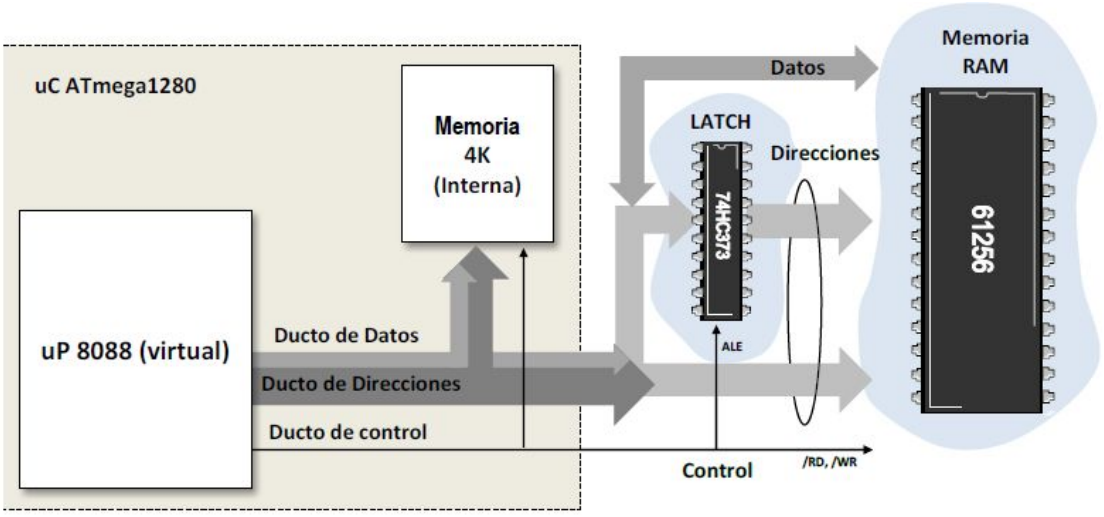


Figura 1. Esquema de Memoria de T-Juino.

Desarrollo:

1) Diseñe e implemente un programa (**lenguaje C + ensamblador**) para probar la expansión de memoria RAM que cumpla con las siguientes requisitos:

a) La prueba de memoria será para el rango de direcciones **2200h** a **FFFFh** que corresponde exclusivamente a la memoria RAM externa.

b) El programa deberá determinar si existe algún problema causado por el **Bus de Datos**, por el **Bus de Direcciones** o por un **Fallo Catastrófico** de la memoria. Indicando la dirección donde se encuentra y la(s) línea(s) afectadas (en el caso de ser un Bus).

Nota: El programa deberá hacer uso de las funciones **peek()** y **poke()** las cuales están implementadas en lenguaje ensamblador (fuera de línea – archivo .asm) y son llamas desde el programa en lenguaje C.

Teoría:

* * * algoritmos de prueba para memoria RAM * * *

Walking One's Algorithm Este algoritmo, permite diagnosticar todos los tipos de defectos eléctricos presentes en una tarjeta de circuito impreso. Este algoritmo solo permite la existencia de un uno en cada vector de test paralelo, de tal manera que el resto son ceros. Para cada vector de test paralelo la posición del 1 ha de ser distinta, por lo que se enviarán tantos vectores como redes haya en el circuito.

El Walking One/Walking Zero Algorithm permite identificar correctamente todos los tipos de cortocircuitos, pues el Walking One es incapaz de detectar los cortocircuitos de tipo AND mientras que el Walking Zero no puede identificar los cortocircuitos de tipo OR. Al usar ambas versiones del mismo algoritmo se logra un diagnóstico completo.

* * * funciones peek y poke() en arquitectura x86 (16bits) * * *

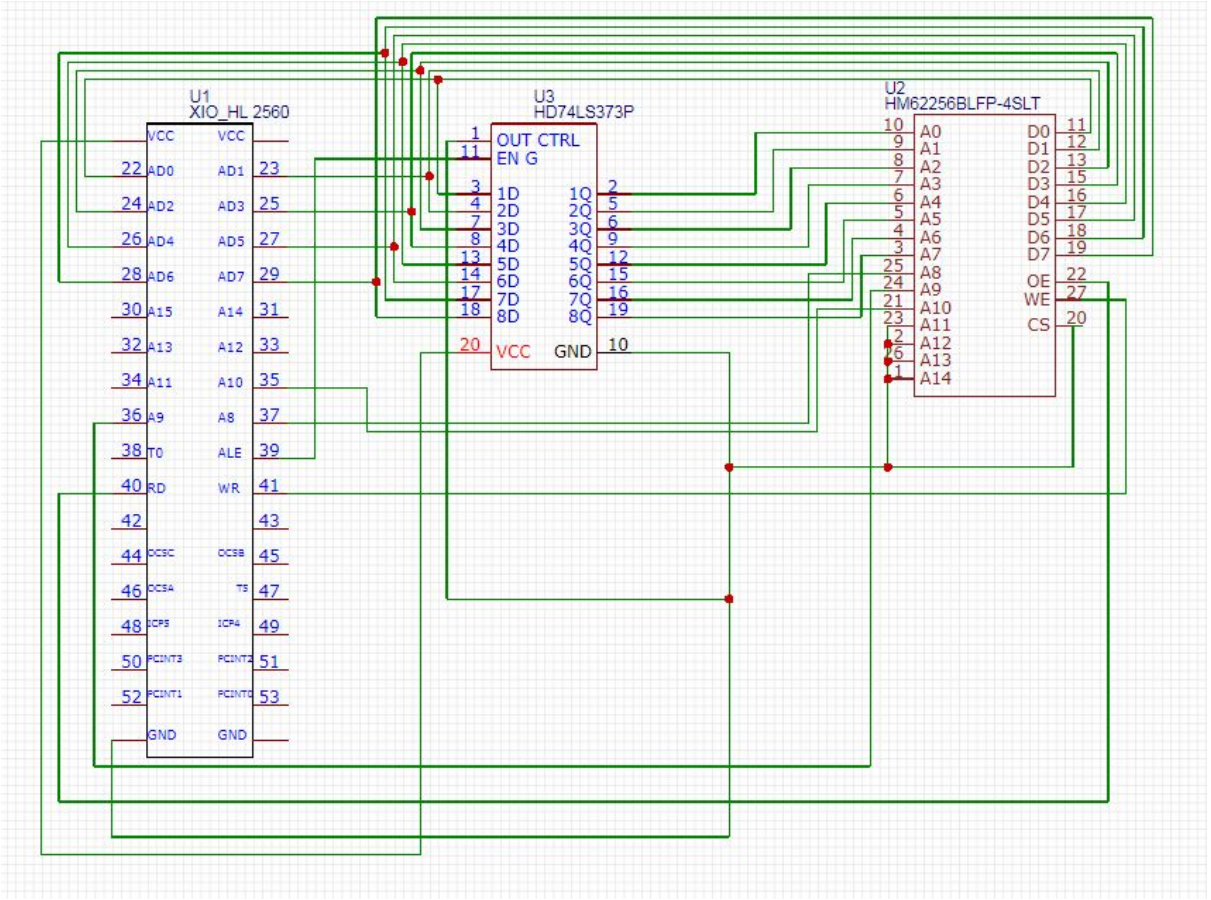
En informática , PEEK y POKE son comandos utilizados en algunos lenguajes de programación de alto nivel para acceder al contenido de una celda de memoria específica a la que hace referencia su dirección de memoria.Estos comandos están particularmente asociados con el lenguaje de programación BASIC , aunque algunos otros lenguajes como Pascal y COMAL también tienen estos comandos. Estos comandos son comparables en sus roles con los punteros en el lenguaje C y algunos otros lenguajes de programación.

Los comandos PEEK y POKE se concibieron en los primeros sistemas informáticos personales para servir a una variedad de propósitos, especialmente para modificar registros de hardware mapeados en memoria especiales para controlar funciones particulares de la computadora, como los periféricos de entrada / salida. Alternativamente, los programadores pueden usar estos comandos para copiar software o incluso para eludir el propósito de un determinado software (por ejemplo, manipular un programa de juego para permitir que el usuario haga trampa). Hoy es inusual controlar la memoria de la computadora a un nivel tan bajo usando un lenguaje de alto nivel como BASIC. Como tal, las nociones de comandos PEEK y POKE generalmente se consideran anticuadas.

Los términos peek y poke se usan a veces coloquialmente en la programación de computadora para referirse al acceso a la memoria en general.

```
void poke (unsigned seg, unsigned offs, int valor);   /* poner palabra valor (16 bits) en seg:offset */
int peek (unsigned seg, unsigned offs);               /* leer la palabra (16 bits) en seg:offset */
```

2) Diseñar el esquemático del circuito de expansión de memoria, y posteriormente alambrarlo, en base a lo sugerido en Figura 1. Una vez alambrado, probarlo con el programa del punto 1.



Conclusiones y Comentarios.

A la hora de probar la memoria en el bus de direcciones surgieron varios errores ya que al haber tantos puts(“”) el intérprete no trabajaba de manera adecuada y por lo tanto fallaba el programa, se decidió separar el código en dos partes una de Datos y otra de Direcciones. Así el programa funcionó de manera correcta.

Bibliografía

https://en.wikipedia.org/wiki/PEEK_and_POKE
https://core.ac.uk/download/pdf/80526415.pdf