PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

MOS 6502

INTEGRANTES (Código PUCP):

Franco Chiroque	20173126
Ivan Galvan	20172361
Franco Ochoa	20171301
Lloyd Castillo	20142280
Christian Prada	20171397
Joaquín Moscoso	20163600
Javier Palacios	20162756
Paolo Galliquio	20173144
Jose Mendoza	20175706

Profesores:

- Romero Gutierrez, Stefano Enrique
- Jimenez Garay, Gabriel Alexandro

ÍNDICE GENERAL

Introducción

El MOS 6502 o MOS Technology 6502 es un microprocesador de 8 bits diseñado por MOS Technology en 1975. Cuando fue introducido fue, con bastante diferencia, la CPU más barata con características completas de mercado, con alrededor de un sexto del precio o menos que las CPU con las que competía de compañías más grandes como Motorola e Intel. Sin embargo, este era más rápido que la mayoría de ellos, y, junto con el Zilog Z80, fueron la chispa de una serie de proyectos de computadores que finalmente darían lugar a la revolución del ordenador personal de finales de los 1970 y principios de los años 1980.

El presente proyecto consiste en la emulación del procesador mencionado a través de un programa codificado en el lenguaje C. Se siguió una metodología funcional y descendente, donde a través de estructuras, arreglos y variables se puede simular los registros, la memoria y las instrucciones del MOS 6502.

Para la creación del proyecto, se usaron las clases y asesorías de los profesores y asistentes del curso de Organización y Arquitectura de Computadoras del ciclo 2020-2.

Link del repositorio: https://github.com/MarceloGal/Proyecto-OAC.git

Descripción de las estructuras utilizadas

Para el caso del presente emulador, MOS 6502, se pensó en definir como parte de las estructuras básicas del mismo, la creación de los registros lo realizamos mediante el tipo de dato uint, en donde representamos el acumulador (1 byte de memoria - uint8_t), registro del índice X (1 byte de memoria - uint8_t), registro de índice Y (1 byte de memoria - uint8_t), el contador del programa (PC) (2 bytes de memoria - uint16_t), el puntero a la pila (SP) (1 byte de memoria - uint8_t), el registro de banderas (1 byte de memoria - uint8_t), el registro de instrucciones (1 byte de memoria - uint8_t) y el registro de direcciones (2 bytes de memoria - uint16_t) englobados en una estructura que se maneja de manera similar al MOS 6502. Además, se utilizará también una estructura para el manejo de la memoria que representará la RAM del procesador en donde se reservó un espacio de memoria de tamaño 256 x 256 bytes el cual representa la memoria del MOS 6502. Finalmente, para mejorar el tiempo de acceso cuando se quiere ejecutar una instrucción se usa como estructura principal un arreglo de puntero a funciones. Esta estructura debe ser, en primer lugar, cargada en una función auxiliar. Luego de haber cargado cada instrucción, esta funciona como una tabla Hash.

Funcionamiento General del Emulador

Antes de realizar la ejecución de algún programa, el emulador asigna a una sección de la memoria las instrucciones disponibles para esta. Esto se realiza a través de arreglos asignados a funciones.

El *bucle* (main) se realiza hasta que el contador de programa apunte a la dirección 0x10, lo cual causaría el rompimiento de este y la finalización del programa para el procesador emulado. Para cada programa, el emulador emite un reporte de los registros, para ello se utiliza la función `print_MOS6502`, que nos muestra el acumulador, registro X, Y ,PC (program counter), SP (stack pointer) y SR (registro de banderas) antes y después de la ejecución con el fin de mostrar el cambio en los registros mencionados.

Etapas del pipeline

Fetch:
Usando el contador del programa, se lee desde la memoria principal el código de la próxima instrucción a ejecutar. El código de esta instrucción es copiado al registro interno de instrucciones del microcontrolador. Para nuestro código, se emula la obtención del código de la instrucción usando la memoria principal como un array y el contador del programa como índice del array. Para luego, copiar ese elemento referenciado a uno de los registros de la estructura "cpu".

Decode:
En esta etapa, se procede a decodificar la instrucción usando el registro interno de instrucciones el cual previamente ya tiene cargado el código operacional. Para nuestra implementación, usamos el arreglo de punteros a funciones podemos acceder a la función de la instrucción solicitada mediante el registro interno de instrucciones. Además, se debe pasar por referencia a esta estructura el "cpu" y la "memoria principal".

Execute:
Esta etapa es opcional dependiendo de las instrucciones. Para casos donde se usa el ALU como en ADC, AND, EOR entre otros, esta etapa sí es ejecutada. Para nuestra implementación, dentro de cada función que emula una instrucción se detalla qué parte es la que hace propiamente execute y cual hace writeback.

Writeback:
En esta etapa, se modifican los registros de microprocesador o el contenido de la memoria principal. Usando la instrucción INX como ejemplo, luego de haber incrementado el valor del registro X, se procede a analizar los *flags.* Según el manual de instrucciones, esta instrucción afecta sólo a *negative flag* y *zero flag* por lo que usando operadores de bits se analiza cada caso, y si cumple, se modifica ese *flag.* Finalmente, se procede a aumentar el contador del programa según sea el caso.

Descripción de Prototipos de las Instrucciones Implementadas

Las funciones que emulan las instrucciones del procesadores comienza ubicando el registro de dirección en el primer *Opcode* de la instrucción. Luego, una vez identificada la función, se realizan las operaciones correspondientes a la instrucción. Después, se revisa si las operaciones anteriores pueden modificar los registros de bandera. Finalmente, el *program counter* salta se aumenta dependiendo el número de argumentos usados por la instrucción (1 + número de argumentos de la instrucción).

Ejm:

```
void ADC_absolute (MOS6502* cpu, MEMORIA* mem){
      // Se ubica el registro de dirección en el primer OPcode
      cpu->addr = cpu->pc+1;
      //Se realizan las operaciones correspondientes
      uint16_t aux = mem->ram[mem->ram[cpu->addr+1]*256 +
mem->ram[cpu->addr]] + cpu->a;
      //BANDERAS
      if(cpu->sr & 0x01) aux += 1;
      if(aux & 0x100) cpu->sr |= 0x01;     //MODIFICAR CARRY
      cpu->a = aux;
      if(cpu->a & 0x80) cpu->sr |= 0x80;   //MODIFICAR NEGATIVE
      else cpu->sr &= 0x7F;
      if(!cpu->a) cpu->sr |= 0x02;         //MODIFICAR ZERO
      else cpu->sr &= 0xFD;
```

```
	//El program counter se aumenta en 2 debido a que solo hay un
argumento
	cpu->pc += 3;
}
```

Programa de prueba

En el archivo de texto “Array-len-code.txt”, se encuentra un programa escrito en OPcodes. Este guarda un arreglo en memoria con una marca de fin: 0x00 (cero). Luego ejecuta un bucle "while" para contar cuántos elementos tiene el arreglo y guarda el resultado en un registro x .
Este programa fue creado para probar funcionalidades como guardar en memoria (STA), cargar datos de memoria (LDA), modificar registros (INX) y uso del branch (BNE).

Es necesario precisar que, el emulador lee opcodes de un archivo de textos; para determinar el valor de estos utilizamos la página https://skilldrick.github.io/easy6502/ .En la siguiente entrega proponemos implementar un analizador léxico que permita identificar las palabras reservadas y sus modos de direccionamiento y que se pueda introducir código propio del 6502 (palabras reservadas) utilizando un archivo de textos.

Bibliografía

WIKIPEDIA
MOS 6502. Consulta: 16 de octubre de 2020.

https://es.wikipedia.org/wiki/MOS_6502

SOSA, Stefano
2020 Apuntes sobre el 6502. Lima.

CARRI, Albertina
2020 Los rubios [videograbación]. Nueva York: Vimeo. Consulta: 16 de octubre del 2020.

http://vimeo.com/44770680

MASSWERK
6502 Instruction Set. Consulta: 16 de octubre de 2020.

https://www.masswerk.at/6502/6502_instruction_set.html#LDA

SKILLDRICK
Easy 6502. Consulta: 7 de octubre de 2020.

https://skilldrick.github.io/easy6502/

Porcentaje	Franco Chiroque	Ivan Galvan	Javier Palacios	Christian Prada	Franco Ochoa	Paolo Galliquio	Lloyd Castillo	José Mendoza	Joaquín Moscoso
Franco Chiroque	100	100	100	100	100	100	100	100	100
Ivan Galvan	100	100	100	100	100	100	100	100	100
Javier Palacios	100	100	100	100	100	100	100	100	100
Christian Prada	100	100	100	100	100	100	100	100	100
Franco Ochoa	100	100	100	100	100	100	100	100	100
Paolo Galliquio	100	100	100	100	100	100	100	100	100
Lloyd Castillo	100	100	100	100	100	100	100	100	100
José Mendoza	100	100	100	100	100	100	100	100	100
Joaquín Moscoso	100	100	100	100	100	100	100	100	100

Columna representa puntaje del evaluado

Fila representa puntaje del evaluador