PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

SECCIÓN INGENIERÍA INFORMÁTICA

Organización y Arquitectura de Computadoras

INFORME 1: EMULADOR MOS 6502

Profesores:

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

Integrantes (Grupo 3):

- 20166046 (H-0681) Oropeza León, Luis Angel Cj
- 20180146 (H-0681) Dueñas Damian, Óscar Eduardo
- 20180303 (H-0681) Llantoy Sanchez, Pilar Maritza
- 20180824 (H-0681) Mejia Padilla, Andrea Adela
- 20181130 (H-0681) Jaimes Agreda, Franccesco Michael
- 20181687 (H-0681) Rafael Artica, César David
- 20182945 (H-0681) Ullilen Falcon, Gianella Antouanet
- 20180196 (H-0682) Alarcón Flores, Dayana del Rosario
- 20180339 (H-0682) Melgarejo Román, Carlos Martín
- 20181309 (H-0682) Sánchez Sánchez, Kimy Solange

2020

I. DESCRIPCIÓN DE LAS ESTRUCTURAS UTILIZADAS

Para emular la arquitectura del procesador 6502, se optó por crear la estructura “MOS6502”, dentro de la cual especificamos los registros; tales como, el *acumulator*, los índices *x* e *y*, el *program counter*, el *stack pointer*, el *status register*, el *instruction register* y el *address register*. Para ello, se tuvo en cuenta el tamaño de cada registro, siendo todos de 8 bits con excepción del *program counter* y el *address register*, que son de 16 bits. Además, se creó la estructura “memory”, conformada por los punteros ram, rom y gpio (Entrada/Salida de Propósito General). Con el objetivo de acceder a dichas estructuras se emplearon las variables globales “cpu” y “mem”, que son punteros a “MOS6502” y “memory”, respectivamente.
Asimismo, se consideró prudente el uso de constantes simbólicas para representar las *flags* y modos de direccionamiento.

II. DESCRIPCIÓN DE LA IMPLEMENTACIÓN PARA CADA UNA DE LAS ETAPAS DEL PIPELINE

El procesador 6502 cuenta con cuatro etapas del Pipeline, *fetch*, *decode*, *execute* y *write back*. Para simular el comportamiento de cada una de ellas se implementaron funciones void.

La primera etapa, *fetch*, usa el *program counter* para acceder a la memoria y saber a qué instrucción hace referencia. Para ello, se asignó al *instruction register* el valor de la memoria en la posición del *program counter.*

La segunda etapa, *decode*, determina los datos necesarios para el programa. Para esto, se asignó al *address register* la dirección del *program counter* aumentado en uno, ya que en esta se encuentra el dato a utilizar.

La tercera etapa, *execute*, ejecuta la operación solicitada mediante la lectura del *instruction register*. Para eso, se realizó una función void en la cual se usa un *switch* de acuerdo al *opcode* de la instrucción cargada.

Finalmente, *write back,* se realiza la escritura en el registro correspondiente y actualiza el *program counter* a la siguiente instrucción a ejecutar. Esta etapa del pipeline se realizó de manera implícita en cada una de las instrucciones.

III. DESCRIPCIÓN DE LOS PROTOTIPOS UTILIZADOS PARA DESARROLLAR LAS INSTRUCCIONES

Para la implementación de las instrucciones, se tomó en cuenta los modos de direccionamiento que cada una de ellas presentaba:
Para aquellas funciones que solo presentaban modo implícito, no se pasó ningún parámetro, ya que en este modo el registro el cual se ve afectado se encuentra sobreentendido en el nombre de la instrucción; por ejemplo *DEY*, que disminuye en uno el valor que se encuentre en el registro *Y*.
Por otro lado, las instrucciones que presentaban solo modo relativo (branch), hacen uso de un parámetro el cual es el desplazamiento.
Asimismo, se implementaron las instrucciones que presentaban más de un modo de direccionamiento haciendo uso de tres parámetros, el primero indica el modo en el cual se ejecutará la instrucción, el segundo indica una dirección o un valor (dependiendo del modo que se ha llamado) y el tercero se usa para los modos absolutos, ya que hacen uso de direcciones

de dieciséis bits. Cabe resaltar que si el modo con el cual se invoca la instrucción no requiere de alguno de los tres parámetros, se escribe el número “2”, el cual se eligió de manera aleatoria teniendo en cuenta que al realizar la conversión a binario de este número no se confunda con algún valor dentro de la función.

IV. DESCRIPCIÓN DE LOS PROTOTIPOS UTILIZADOS PARA DESARROLLAR LOS MODOS DE DIRECCIONAMIENTO

Como se mencionó previamente, para los modos de direccionamiento se usaron constantes simbólicas, asignándole a cada modo un número dentro del rango de 0 a 12. Esto con la finalidad de que en cada instrucción (si se amerita) se indique el modo en el primer parámetro. Hecho esto, dentro de la implementación de cada instrucción se realiza un *switch* de acuerdo al modo invocado y así se ejecutan las operaciones necesarias.

V. CODIFICAR UN PEQUEÑO PROGRAMA DE PRUEBA PARA VERIFICAR EL FUNCIONAMIENTO DE SU EMULADOR

En primer lugar, se inicializan los registros, la memoria y el cpu, con la función *init* y *reset_cpu.*
A continuación, para el programa de prueba es necesario generar el volcado en hexadecimal. Para ello, se desarrolló un programa en C++ que toma un archivo ASM para MOS6502, y basándose en el conjunto de instrucciones de este procesador genera un archivo de texto con el volcado en hexadecimal. Dicho archivo es leído y almacenado en la memoria del procesador. Cabe resaltar que se tiene un contador que guarda la cantidad de bytes que se guardaron en la memoria; así cada vez que el *program counter* se actualice, se sabrá si se ha llegado al fin del programa.
Una vez realizado este proceso, se ejecuta el programa que se encuentra guardado en la memoria. Esto invoca a las funciones fetch, decode y execute iterativamente hasta que el *program counter* llegue al valor del contador mencionado previamente.
Por último, con la finalidad de validar que las instrucciones funcionan correctamente, se decidió imprimir los valores de los registros luego de ejecutarlas.

FUNCIONES DE APOYO:

Prototipos de funciones para las banderas

- **set_flag(uint8_t flag, uint8_t status):** Setea en “1” ó “0” la bandera que se indique
- **flagZ(uint8_t result):** Setea la bandera Z en el valor que se indique como parámetro
- **flagN(uint8_t result):** Setea la bandera N en el valor que se indique como parámetro
- **flagC(uint16_t):** Setea la bandera C en el valor que se indique como parámetro
- **flagD(uint8_t):** Setea la bandera D en el valor que se indique como parámetro
- **flagV(uint8_t operand1,uint8_t operand2,uint16_t result):** Setea la bandera V

Prototipos de funciones auxiliares

- **printf_binary(int n, int i):** Imprime el número que se indica en base dos,con una precisión “i” de bits.
- **print_caracter(char c, int n):** Imprime el carácter “c” n veces, solo sirve para formato de reportes.
- **printf_registers():** Imprime todos los registros del procesador.
- **loadToMemory(uint16_t init_pc):** Lee el archivo .txt, devuelve la cantidad de bytes que contiene el programa.
- **init():** Inicializa los registros y la memoria.

- **reset_cpu():** Inicializa el cpu.
- **print_memory(uint16_t inicio, int n):** Imprime los valores, en el terminal, que se encuentran en memoria, desde inicio, que es una dirección, hasta el número “n” de líneas indicado teniendo en cuenta que cada línea contiene 16 espacios de memoria.
- **print_memory_report(char *name):** Imprime toda la memoria en un archivo .txt de nombre “name” que se indique.

VI. LINK DE REPOSITORIO

https://github.com/angelleonle/emuladorMOS6502-Equipo-03-OAC

VII. ANEXOS

TABLA DE PARTICIPACIÓN		
Código	**Participante**	**Porcentaje de participación (%)**
20166046	Oropeza León, Luis Angel Cj	100
20180146	Dueñas Damian, Óscar Eduardo	100
20180303	Llantoy Sanchez, Pilar Maritza	100
20180824	Mejia Padilla, Andrea Adela	100
20181130	Jaimes Agreda, Franccesco Michael	100
20181687	Rafael Artica, César David	100
20182945	Ullilen Falcon, Gianella Antouanet	100
20180196	Alarcón Flores, Dayana del Rosario	100
20180339	Melgarejo Román, Carlos Martín	100
20181309	Sánchez Sánchez, Kimy Solange	100