Facultad de Ciencias e Ingeniería, Pontificia Universidad Católica del Perú

IEE240: ORGANIZACIÓN Y ARQUITECTURA DE COMPUTADORAS

Informe N°1

Integrantes	**Código**
Toro Vela Carlos Santos	20171878
Huaylla Mendoza, Sandro	20171364
Acosta Gonzales, Alonso Oswaldo	20170809
Davila Diaz Fabricio, Sebastián	20181458
Bartra Quiste, Ricardo Alexander	20176243
Tarazona Pariamachi, Alfredo	20181348
Vicente Yupanqui, Diego David	20171582
Córdova Rivero, Iván Sebastián	20181923
Aragón Yauris, Joaquín Arturo	20181252

ÍNDICE

INTRODUCCIÓN:

El MOS 6502 es un microprocesador de 8 bits, que fue diseñado por MOS Technology en 1975. Fue, en su momento, uno de los procesadores más rápidos del mercado y con un precio muy accesible. Debido a eso, grandes compañías se interesaron en promover su uso para el desarrollo de sus intereses propios. Uno de estos, probablemente el más conocido, fue el diseño de la videoconsola Atari 2600. Años después, el éxito relevante en el mercado de este microprocesador incentivó su uso para fines académicos, como introducción al lenguaje ensamblador o en algunos sistemas integrados. En la actualidad, se siguen fabricando para los fines descritos anteriormente.

Para el presente trabajo utilizaremos este microprocesador, incluyendo su arquitectura, sus ventajas y desventajas, para emularlo en un lenguaje de alto nivel y a su vez realizar diversas aplicaciones con este.

OBJETIVO:

El objetivo del programa es presentar la creación, en funcionalidad, del emulador 6502 en lenguaje C . A su vez estas instrucciones servirán a futuro para poder presentar un juego diseñado solamente con la arquitectura 6502.

DESCRIPCIÓN DE LAS ESTRUCTURAS UTILIZADAS:

Para poder emular el funcionamiento del 6502 utilizaremos dos estructuras principales las cuales son:

```
struct CPU6502 {
    uint8_t a;
    uint8_t x;
    uint8_t y;

    uint8_t pch;
    uint8_t pcl;

    uint8_t sp;
    uint8_t sr;

    uint8_t pcu;
};
```

La estructura principal es **el procesador 8 bits**, este posee diferentes registros del mismo tamaño que describiremos a continuación.

a: Registro acumulador (8 bits).

x: Registro índice X (8 bits).

y: Registro índice Y (8 bits).

pch: Byte más significativo del Program Counter (8 bits)

pcl: Byte menos significativo del P.C. (8 bits)

sp: Stack Pointer (8 bits)

sr: Status Register, registro de banderas (8 bits)

pcu: Flag del P.C. , se activa después de haberse modificado.

```
struct MEM6502 {
    uint8_t **ram;
};
```

Nuestra segunda y última estructura es la **memoria**, la cual representaremos como una matriz de tamaño 256x256 de manera dinámica. Con esto hacemos referencia a las 256 páginas con 256 bytes cada una. Además, inicializamos toda la memoria en 0.

ram: puntero hacia la memoria.

DESCRIPCIÓN DE LA IMPLEMENTACIÓN PARA CADA UNA DE LAS ETAPAS DEL PIPELINE:

Nuestra implementación cuenta únicamente con 2 etapas, **FETCH** (IF) y **EXECUTE** (EXE), estas se encargan de realizar todo el proceso requerido para procesar cada una de las instrucciones. Se diseñó el programa de tal manera que, no fue necesario realizar una etapa previa de **DECODE** (ID), y posterior de **WRITE-BACK** (WB) ya que implícitamente y por como está diseñado el programa, esta labor lo realiza la función **Execute**. A continuación se realizará una breve descripción de cómo fueron implementadas cada una de las etapas del pipeline.

FETCH: Se encarga de retornar la posición donde se encuentra el program counter, que a su vez, coincide con la ubicación que contiene el OPCODE de la siguiente instrucción a ser ejecutada por el programa.

```
int fetch()
{
    return mem.ram[cpu.pch][cpu.pcl];
}
```

Retorna el índice del opcode:

EXECUTE: Esta función recibe como parámetro el índice del opcode de la instrucción a ejecutar, esto se debe a que se creó un arreglo con punteros a funciones donde cada instrucción tiene como índice su opcode correspondiente.

Lo primero que se realiza es asignar 0 al flag que controla si el PC ha sido modificado, luego de eso, con el opcode de la instrucción a ejecutar, se obtiene de un arreglo la cantidad de bytes que contiene dicha instrucción, para posteriormente modificar el PC. Luego, a través del opcode como índice, hacemos que "func" apunte a la función solicitada, y se ejecuta. Finalmente, si el Program counter no ha sido modificado en la instrucción, se le suma a este el número de bytes que posee la instrucción que fue ejecutada. El uso del flag "pcu" es importante para el caso de branches, en donde el program counter es modificado durante la ejecución de la instrucción,y este ya no debe cambiar posteriormente de haber culminado.

```
void execute(int op_ind)
{
    cpu.pcu = 0;
    uint8_t num = opcode_bytes[op_ind];
    void (*func) (void) = opcode_func[op_ind];
    func();
    updatepc(num);
}
```

```
void updatepc(uint8_t numbytes)
{
    if (!cpu.pcu) {
        if ((int)cpu.pcl + numbytes > 0xff) cpu.pch += 1;
        cpu.pcl += numbytes;
    }
}
```

DESCRIPCIÓN DE LOS PROTOTIPOS UTILIZADOS PARA DESARROLLAR LAS INSTRUCCIONES:

Todas nuestras instrucciones son de tipo void, y no reciben parámetros. Estos trabajan de esta manera debido a que, se utiliza la función **"get_arg(n)"** para obtener el byte que necesitamos a partir del program counter, el cual se encuentra en la memoria. Esta fue cargada al inicio del programa, obteniendo la información del archivo binario que fue generado por el acme al momento de compilar.

El acme, de manera complementaria, es un ensamblador cruzado gratuito que permite producir código máquina para diferentes procesadores a partir de otro procesador, entre ellos el 6502, este toma un lenguaje ensamblador como entrada y da el lenguaje máquina solicitado como salida. Además, permite incluir archivos binarios mientras realiza el proceso de ensamblaje, el cual, usamos para la elaboración de nuestra emulación.

DESCRIPCIÓN DE LOS PROTOTIPOS UTILIZADOS PARA DESARROLLAR LOS MODOS DE DIRECCIONAMIENTO:

Para el desarrollo de los modos de direccionamiento de las instrucciones, se decidió crear una función para cada caso, así tendremos, por ejemplo, 8 funciones que hacen referencia a LDA, cada uno con su respectivo modo de direccionamiento. Cada una de estas posee una ubicación diferente en nuestro arreglo de punteros a funciones, en la posición de su opcode correspondiente como índice. Así se facilitará el proceso de trabajar con los diversos modos de direccionamiento que posee el 6502.

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

Programa: Debido a que el 6502 no cuenta con una instrucción de multiplicación, se ha implementado una con las instrucciones básicas que esta posee. Importante notar que el resultado se guarda en memoria en hexadecimal, en este caso en la posición $00, además, en el registro A.

```
9   factor1: !by 4 ;declaramos el operando 1, con el valor 4
10  factor2: !by 3 ;declaramos el operando 2, con el valor 3
11  ;los numeros se guardan en hexadecimal en la memoria
12  jsr multiplicar ;subrutina de multiplicacion
13                  ;el resultado estara la direccion $00
14  jmp exit
15
16  multiplicar:
17      ;el contador estará en la pila
18      lda #$00
19  sumar:
20      cmp factor2
21      beq exit
22      adc #$01
23      pha ; pusheo el contador
24      lda $00 ;en $00 se guarda la suma
25      adc factor1
26      sta $00
27      pla ; pop del contador
28      jmp sumar
29  rts
30
31  exit:
32      lda $00
33      !byte $04 ;imprimimos en pantalla los registro del CPU
34      !byte $02 ;imprimimos la memoria RAM
35      ;fin programa
```

```
A = 0c          C Z I D B V N
X = 00          1 1 0 0 0 1 0
Y = 00
```

REPOSITORIO DE GITHUB:

El repositorio se encuentra en público, por lo que no será necesario pertenecer a este para poder visualizarlo y descargarlo : https://github.com/PUCP-INF/mos6502-emulator

TABLA CON EL PORCENTAJE DE PARTICIPACIÓN:

PARTICIPANTES	PORCENTAJE PARTICIPACIÓN
Toro Vela , Carlos Santos	100%
Huaylla Mendoza,Sandro	100%
Acosta Gonzales, Alonso Oswaldo	100%
Bartra Quiste, Ricardo Alexander	100%
Dávila Díaz, Fabricio Sebastián	100%
Tarazona Pariamachi, Alfredo	100%
Aragón Yauris, Joaquín Arturo	100%
Córdova Rivero, Iván Sebastián	100%
Vicente Yupanqui, Diego David	100%