Universidad Industrial de Santander

Arquitectura de computadores 2024-2

# Proyecto 5: Arquitectura de la computadora

Nucleo Digital

El proyecto 5 de Nand2Tetris, llamado arquitectura de la computadora, consiste en combinar la ALU y RAM construidas en proyectos anteriores para construir la plataforma de hardware Hack completa. Lo que resultaría en una computadora de propósito general que pueda ejecutar programas escritos en lenguaje de máquina Hack.

**Memory:**

El propósito del chip Memoria es manejar 3 componentes:

- RAM16K: Memoria principal. Este chip se realizó en el proyecto 3.
- Screen: Espacio de memoria reservado para la presentación en pantalla. Este chip es proporcionado en este proyecto.
- Keyboard: Espacio de memoria para las entradas de teclado. Este chip es proporcionado en este proyecto.

Las entradas de este chip son:

- in[16]: Son los datos que se desean almacenar.
- load: Indicador de si se deben almacenar los datos ingresados.
- adress[15]: Dirección de 15 bits que determina donde se deben almacenar/leer los datos:
    - Si adress[14] = 0: Se debe acceder a RAM16K.
    - Si adress[14] = 1:
        - Si adress[13] = 0: Se debe acceder a pantalla.
        - Si adress[13] = 1: Se debe acceder al teclado.

La salida out[16] son los datos leídos desde la dirección indicada.

Para realizar el código de este chip, se dividió en 3 partes:

1. A partir de lo que determinan los bits 14 y 13 de la dirección, se utilizaron los chips And16 y Not16 para determinar si se debe acceder a RAM16K, a Screen o a Keyboard.

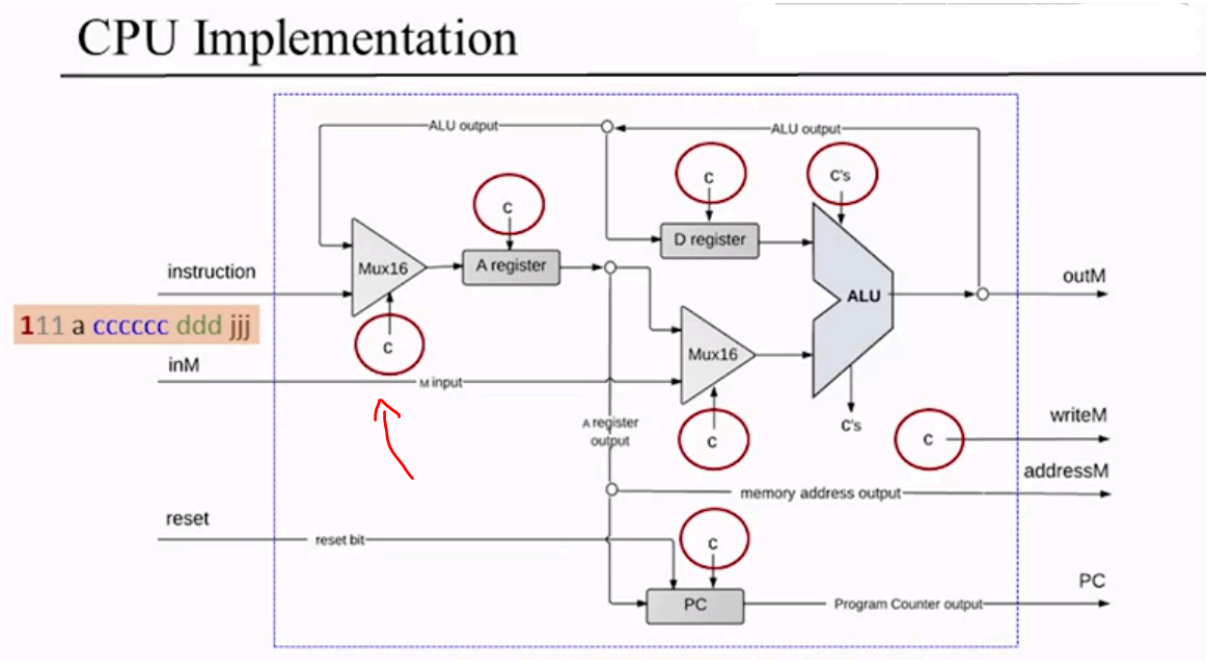
2. Posteriormente se inicializar los 3 chips (RAM16K, Screen y Keyboard), trayendo sus respectivas salidas. Tanto RAM16K como Screen requieren de la entrada “load”.

3. Finalmente a través de multiplexores (Mux16) se selecciona cual debe ser la salida de Memoria (si la de RAM16K, Screen o Keyboard). Los selectores serán las salidas de la parte 1 de este código.

```
CHIP Memory {
    IN in[16], load, address[15];
    OUT out[16];

    PARTS:
    And(a=address[14], b=address[13], out=teclado);
    Not(in=teclado, out=noteclado);
    And(a=address[14], b=noteclado, out=pantalla);

    And(a=load, b=noteclado, out=loadRam);
    And(a=load, b=pantalla, out=loadPantalla);

    RAM16K(in=in, address=address[0..13], load=loadRam,
out=salRam);
    Screen(in=in, address=address[0..12], load=loadPantalla,
out=salPantalla);

    Keyboard(out=salTeclado);

    Mux16(a=salRam, b=salPantalla, sel=pantalla,
out=salRamPantalla);
    Mux16(a=salRamPantalla, b=salTeclado, sel=teclado,
out=out);
}
```

**CPU:**

En este código HDL se escriben las partes (PARTS) se basan en el esquema en el que la ALU es el componente central. Es necesario realizar varias operaciones lógicas y aritméticas sobre los datos, específicamente sobre las instrucciones, que están codificadas de una manera particular para controlar los diversos chips, como los registros A y D, la escritura en memoria y la ALU misma. Es por ello que utilizamos compuertas lógicas como AND y OR, las cuales nos permiten interpretar y manipular los bits de las instrucciones para coordinar el flujo de datos entre los distintos componentes de la CPU.




**Computer:**

```
CHIP Computer {
    IN reset;

    PARTS:
    ROM32K(address=pc, out=instruction);
    CPU(inM=memOut, instruction=instruction, reset=reset,
    outM=outM, writeM=writeM, addressM=addressM, pc=pc);
    Memory(in=outM, load=writeM, address=addressM, out=memOut);
}
```

## Análisis del Código:

1. **Definición del Chip**: `CHIP Computer` define un nuevo chip llamado `Computer` con una entrada llamada `reset`, que se utiliza para reiniciar la ejecución del programa.
2. **ROM32K**: Este componente está correctamente definido para proporcionar las instrucciones del programa. Se usa `address=pc` para indicar que el contador de programa (`pc`) apunta a la dirección de la instrucción actual, y `out=instruction` para enviar esa instrucción a la `CPU`.
3. **CPU**: La instancia de `CPU` parece bien configurada. Recibe la entrada de memoria (`inM=memOut`), la instrucción (`instruction`), y el `reset`, así como las salidas necesarias para escribir en la memoria (`outM`, `writeM`, `addressM`, `pc`).
4. **Memory**: La memoria se ha configurado para recibir datos de salida de la CPU (`in=outM`), controlar la carga con `load=writeM`, y permitir la lectura en una dirección específica (`address=addressM`). La salida es `memOut`, que se usa como entrada en la CPU.

## Conclusión

El código está bien estructurado y sigue la lógica de la arquitectura Hack. Asegúrate de que los componentes `ROM32K`, `CPU`, y `Memory` estén implementados correctamente en tu entorno de trabajo para que este código funcione como se espera. Si necesitas pruebas adicionales o ejemplos de uso, puedes consultar el sitio oficial de nand2tetris para más información y recursos.

**Bibliografia:**

**Dunfelt, E. (n.d.). *Nand2tetris*. GitHub. Retrieved from https://github.com/edunfelt/nand2tetris**