QSim: Simulador de arquitecturas Q

Susana Rosito Tatiana Molinari

9 de noviembre de 2013

Índice

Resumen

Este trabajo...

Parte I
Contexto

1. Sobre la materia Organización de Computadoras(susi)

2. Conceptos importantes

2.1. Enfoque de Von Neumann

[1]Este enfoque generaliza la organización de las computadoras distinguiendo en tres partes interconectadas: La CPU (con la unidad aritmético-lógica o ALU y la unidad de control) la memoria, y un módulo de entrada/salida. (con un bus de sistema que proporciona un medio de transporte de los datos entre las distintas partes)[2].

Con la propuesta de este modelo Von Neumann incorpora el concepto de **programa almacenado**contiene un conjunto de instrucciones que podían ser almacenadas en memoria, o sea, un programa que detalla la computación del mismo[3]. El programa se codifica de cierta manera para que pueda ser almacenado y posteriormente ejecutado quizás múltiples veces. Ademas puede ser 'recordado' de esta manera se separa en el mecanismo de ejecución el hadware de la logica de computo o instrucciones (software).[4]

Este tipo del diseño que permite un programa almacenado también da la posibilidad de que los programas sean modificados ellos mismos durante su ejecución[5]. Por ejemplo un programa podría modificar o incrementar [6] las direcciones de memoria que tenga en algunas instrucciones y luego volver a ejecutar dichas instrucciones con el fin de procesar celdas diferentes de memoria cada vez. Esta característica es potente y a la vez peligrosa[7] pues las modificaciones en los programas podía ser algo perjudicial, por accidente o por diseño.

[1](Un poco de historia) El matemático John Von Neumann propone en el año...

[2]La interconexión es llevada a cabo por un un bus de sistema que proporciona un medio de transporte de los datos entre las distintas partes

[3]no entiendo esta frase. la puse yo?

[4]Con esta idea, el programa se codifica de cierta manera para que pueda ser almacenado en memoria principal y posteriormente ejecutado quizás múltiples veces. De esta manera, la lógica del programa puede ser "recordada" y el programa toma un valor mayor, a diferencia de lo que ocurría hasta entonces, donde el programa se reflejaba en un conjunto de configuraciones de cables aplicadas a los equipos. Esto implica una separación entre el mecanismo de ejecución (el hadware) y la lógica de computo o instrucciones (el software).

[5]la ejecución de las instrucciones modifique el código máquina del mismo u otro programa

[6]las referencias a

[7]pero presenta un alto riesgo

2.2. Organización de la computadora

La CPU (Unidad Central de Procesamiento del inglés: Central Processing Unit), es el componente principal y el encargado ejecutar los programas y procesar los datos. La CPU contiene otros componentes de importancia tales como la **Unidad de Control**, PC[8], IR[9], Registros[10] y la ALU[11].

La Unidad de Control dirige el ciclo de ejecución de cada instrucción, pidiendo la lectura de celdas de memoria donde esta alojada, decodificándola y ejecutándola luego en colaboración con los otros componentes del sistema: si es una operación lógica o aritmética le ordena a la ALU su ejecución, si es de movimiento de datos colabora con la memoria ó el módulo de Entrada/Salida.

El contador de programa (en inglés *Program Counter* o PC) es un registro que indica la posición de memoria donde estará la siguiente instrucción que debe ejecutarse. Este registro se incrementa luego de cada ciclo de instrucción[12].

El registro de instrucción contiene [13] la instrucción actual una vez que la misma es leída de memoria para luego decodificarla y ejecutarla.

Registros: son elementos de una memoria de alta velocidad y poca capacidad que permite guardar y acceder a valores generalmente muy usados, por ejemplo, operandos en operaciones matemáticas. Hay diferentes tipos de registros, algunos [14] pueden guardar tanto datos como direcciones de memoria.

La Unidad Aritmético-Lógica, recibe su nombre de las siglas en inglés de *Arithmetic and Logic Unit*. La ALU es un circuito digital que lleva a cabo operaciones aritméticas (suma, resta, multiplicación, división) y las operaciones lógicas (Negacion, Y, O, O exclusivo)[15], entre dos cadenas binarias que son interpretadas como números [16].

La memoria es un conjunto de celdas numeradas. La numeración de cada celda la identifica inequívocamente por lo cual a esta numeración se le llama dirección. En cada celda de la memoria se pueden almacenar datos o instrucciones en forma de cadenas binarias y este contenido puede leerse y modificarse. En la memoria es donde se alojan los programas que luego serán ejecutados.

[8] el contador de programa (PC o *Program Counter*)
[9] el registro de instrucción (IR - *Instruction Register*)
[10] otros registros de uso específico y general
[11] Unidad Aritmético-Lógica (ALU)
[12] Luego de completar el ciclo de ejecución de una instrucción, el PC se incrementa en función de la cantidad de celdas que ocupa el código máquina de esta
[13] el código máquina de
[14] El diseño de cada arquitectura ofrece un conjunto diferente de registros de uso general para ser usados en los programas. Estos registros son elementos de memoria de alta velocidad y poca capacidad que pueden ser utilizados como variables en los programas. Es importante marcar que
[15] como la negación, disyunción, conjunción, etc.
[16] o valores lógicos

Buses: son los encargados de[17] transferir los datos entre los componentes de la computadora.La unidad de control al pedir un contenido de una dirección de memoria lo hace a través del bus, y similarmente mismo cuando desea escribir en memoria.

2.3. Ejecución de un programa

La función de una computadora es la ejecución de programas. Los programas se encuentran almacenados en memoria y consisten en una sucesión de instrucciones que posee orden. La CPU es quien se encarga de ejecutar dichas instrucciones a través de un ciclo denominado ciclo instrucciones. Para ser almacenadas en memoria, las instrucciones deben codificarse en cadenas binarias (secuencias de ceros y unos) que no son legibles para las personas pero que la UC puede interpretar y traducir en acciones. Por eso para saber de qué instrucción se trata, y cuales son los valores o celdas de memoria que debería consultar o usar, la cpu utiliza la UC, que tomando bit por bit interpreta la cadena binaria y verificando los códigos únicos de cada instrucción o modos de direccionamiento, sabe qué instrucción y con qué valores debería ejecutar. La ejecución de instrucciones se divide en tres etapas importantes, fech - decode - execute.

Al principio de cada ciclo de ejecución, durante el fech de instrucción la CPU busca una instrucción que se encuentra en alguna parte de la memoria. Para saber donde esta dicha instrucción la CPU contiene un registro llamado contador de programa (PC), que tiene la dirección de la próxima instrucción a buscar. La CPU a través del bus lee la instrucción, y luego incrementa el valor contenido en PC; así podrá buscar la siguiente instrucción en la secuencia luego de terminar con la actual. La instrucción leida que está en la forma de cadena binaria se carga dentro de otro registro de la CPU, llamado registro de instrucción (IR).

Durante la decodificacion la UC determina que significa la cadena binaria.

Finalmente al saber de qué instrucción se trata la CPU ejecuta la instrucción, es decir, realiza lo que la instrucción dice que debe hacer con sus argumentos, modificando la memoria o los registros como resultado final y el ciclo vuelve a comenzar hasta que el programa termine.

3. Arquitecturas Q

3.1. Características generales(susi)

Aqui voy yo!!

3.2. Modos de direccionamiento

1. **Inmediato** Codigo de modo de direccionamiento: 000000 Este modo de direccionamiento es un valor que será utilizado como modo origen pero nunca como modo destino ya que en el no pueden guardarse datos.

   Ejemplos: **0x0000** representa el modo de direccionamiento inmediato cuyo valor es cero. **0x000F** representa el modo de direccionamiento inmediato cuyo valor es 15.

[17] El bus de sistema es el encargado de

2. **Directo** Codigo de modo de direccionamiento: 001000 Este modo de direccionamiento representa una direccion de memoria o de puertos que será utilizado como modo origen pero nunca como modo destino ya que en el no pueden guardarse datos.

   Ejemplos: [**0x0000**] representa el modo de direccionamiento directo cuyo valor se encuentra en la celda de memoria que identifica uniquevocamente a la dirección **0x0000**, es decir 0. [**0x000F**] representa el modo de direccionamiento directo cuyo valor se encuentra en la celda de memoria que identifica uniquevocamente a la dirección **0x000F**, es decir 15.

3. **Indirecto** Codigo de modo de direccionamiento: 011000 Este modo de direccionamiento representa una direccion de memoria o de puertos que será utilizado como modo origen pero nunca como modo destino ya que en el no pueden guardarse datos.

   Ejemplos: [[**0x0000**]] representa el modo de direccionamiento indirecto cuyo valor se encuentra en la celda de memoria cuya dirección esta guardada como dato en la celda de memoria que identifica uniquevocamente a la dirección **0x0000**, es decir 0. [[**0x000F**]] representa el modo de direccionamiento indirecto cuyo valor se encuentra en la celda de memoria cuya dirección esta guardada como dato en la celda de memoria que identifica uniquevocamente a la dirección **0x000F**, es decir 15.

4. **Registro** Codigo de modo de direccionamiento: 100rrr Donde rrr.[es] una cadena binaria que puede tomar los valores desde 000 hasta 111 para identificar según el numero decimal que representen los diferentes ocho registros R0..R7.

   Ejemplos: **R0** representa el modo de direccionamiento registro cuyo valor se encuentra en el registro **R0**. **R7** representa el modo de direccionamiento registro cuyo valor se encuentra en el registro **R7**.

5. **RegistroIndirecto** Codigo de modo de direccionamiento: 110rrr Donde rrr.[es] una cadena binaria que puede tomar los valores desde 000 hasta 111 para identificar según el numero decimal que representen los diferentes ocho registros R0..R7.

   Ejemplos: [**R0**] representa el modo de direccionamiento registro cuyo valor se encuentra en la direccion de memoria que esta guardada como dato el registro **R0**. [**R7**] representa el modo de direccionamiento registro cuyo valor se encuentra en la direccion de memoria que esta guardada como dato el registro **R7**.

3.3. Repertorio de instrucciones

3.3.1. Instrucciones de 2 operandos

El formato de instruccion de dos operandos es el siguiente:

CodOp + Modo destino + Modo origen + Destino + Origen (4 bits) (6 bits) (6 bits) (16 bits) (16 bits)

Donde el operando destino no puede ser un modo de direccionamiento Inmediato, ya que en la direccion de memoria o registro que describa dicho operando es donde se guardará el resultado de la operación.

1. **MUL destino, origen** Código de operación: 0000 Esta instrucción describe la multiplicación entre los datos de los dos operandos. Esta operación es la única que cuyo resultado puede ser 32 bits, que son dos celdas de memoria en hexadecimal, por lo que los primeros 16 bits, es decir, la primer mitad, es guardada en el registro R7 y la segunda en el operando destino.

2. **ADD destino, origen** Código de operación: 0010 Esta instrucción describe la suma entre los datos de los dos operandos. El resultado de la ejecución de la suma es guardado en el operando destino.

3. **SUB destino, origen** Código de operación: 0011 Esta instrucción describe la resta entre los datos de los dos operandos. El resultado de la ejecución de dicha resta es guardado en el operando destino.

4. **DIV destino, origen** Código de operación: 0111 Esta instrucción describe la división entre el dato en el operando destino como dividendo y el dato en el operando origen como divisor. El resultado de la ejecución de la división es guardado en el operando destino.

5. **MOV destino, origen** Código de operación: 0001 Esta instrucción describe la copia de datos del dato alojado en el operando origen al operando destino. El resultado de la ejecución del MOV es el dato guardado en el operando origen ahora también guardado en el operando destino.

6. **AND destino, origen** Código de operación: 0100 Esta instrucción describe la operación logica ƨ"bit a bit entre los datos de los dos operandos. El resultado de la ejecución de esta operación es guardado en el operando destino.

7. **CMP destino, origen** Código de operación: 0110 Esta instrucción describe la comparación bit a bit entre los datos de los dos operandos. El resultado de esta operación es solamente la actualización de flags en la cpu.

8. **OR destino, origen** Código de operación: 0101 Esta instrucción describe la operación logica .º"bit a bit entre los datos de los dos operandos. El resultado de la ejecución de esta operación es guardado en el operando destino.

3.3.2. Instrucciones de 1 operando origen

El formato de instruccion de un operando origen es el siguiente:
CodOp + relleno + Modo origen + Origen (4 bits) (000000) (6 bits) (16 bits)

1. **CALL origen** Código de operación: 1011 El efecto del CALL es guardar la direccion de memoria en la celda de la dirección que se encuentra guardadad en el SP (Stack pointer) aumentar el SP y guardar en el PC (Program Counter) el dato que se encuentra guardado en el operando origen ya que describe el llamado a una subrutina que comienza en la celda de memoria cuya direccion esta guardada en el operando origen.

2. **JMP origen** Código de operación: 0110 El efecto del JMP es cambiar el PC (Program Counter) por el dato que esta guardado en el operando origen ya que esta operación describe el salto a otra parte de la memoria para continuar con la ejecución del programa.

3.3.3. Instrucciones de 1 operando destino

El formato de instrucción de un operando destino es el siguiente:
CodOp + Modo origen + relleno + Origen (4 bits) (6 bits) (000000) (16 bits)

1. **NOT destino** Código de operación: 1001 Esta instrucción describe la operación logica "negación"bit a bit en el datos del operando destino. El resultado de la ejecución de esta operación es guardado en la misma celda o registro de donde es leído el dato inicialmente.

3.3.4. Instrucciones sin operandos

El formato de instrucción sin operandos es el siguiente:
CodOp + relleno (4 bits) (000000000000)

1. **RET** Código de operación: 0110 El efecto del ret es cambiar el pc por el dato que esta guardado en la celda de memoria que se encuentra en el SP (Stack pointer) y decrementar el SP ya que describe la finalización de la ejecución de una subrutina y la ejecución del resto del programa.

3.3.5. Instrucciones de salto condicional

El formato de instrucción de salto condicional es el siguiente es el siguiente:
prefijo + CodOp + desplazamiento (1111) (4 bits) (8 bits)
El efecto de cualquier salto condicional es aumentar el PC (Program Counter) en la cantidad de celdas que indique el desplazamiento si sólo la condición que cada salto condicional tiene da como resultado 1, lo cual es interpretado como verdadero.

1. **JE desplazamiento** Código de operación: 0001 La condición del salto es que el flag **Z** (Cero) sea 1, es decir la ultima operación matemática dió como resultado el número cero.

2. **JNE desplazamiento** Código de operación: 1001 La condición del salto es que el flag **Z** (Cero) sea 0, es decir la ultima operación matemática no dió como resultado el número cero.

3. **JLE desplazamiento** Código de operación: 0010 La condición del salto es el resultado de la siguiente operación lógica **Z OR (N XOR V)**, es decir la ultima operación matemática es menor o igual con signo.

4. **JG desplazamiento** Código de operación: 1010 La condición del salto es el resultado de la siguiente operación lógica **NOT (Z OR (N XOR V))**, es decir la ultima operación matemática es mayor con signo.

5. **JL desplazamiento** Código de operación: 0011 La condición del salto es el resultado de la siguiente operación lógica **N XOR V**, es decir la ultima operación matemática es menor con signo.
6. **JGE desplazamiento** Código de operación: 1011 La condición del salto es el resultado de la siguiente operación lógica **NOT (N XOR V)**, es decir la ultima operación matemática es mayor o igual con signo.
7. **JLEU desplazamiento** Código de operación: 0100 La condición del salto es el resultado de la siguiente operación lógica **C OR Z**, es decir la ultima operación matemática es menor o igual sin signo.
8. **JGU desplazamiento** Código de operación: 1100 La condición del salto es el resultado de la siguiente operación lógica **NOT (C OR Z)**, es decir la ultima operación matemática es mayor sin signo.
9. **JCS desplazamiento** Código de operación: 0101 La condición del salto es que el flag **C** sea 1, es decir la ultima operación matemática es menor sin signo.
10. **JNEG desplazamiento** Código de operación: 0101 La condición del salto es que el flag **N** sea 1, es decir si el último resultado de una operación dio negativo.
11. **JVS desplazamiento** Código de operación: 0111 La condición del salto es que el flag **V** sea 1, es decir si el último resultado de una operación dio overflow.

3.4. Versiones de la arquitectura (SUSI)

hacer un grafico copado donde se vea las capas de cebolla

Las versiones de la arquitectura Q están pensadas para incorporar funcionalidades de manera que la curva de aprendizaje sea adecuada para los alumnos, siendo paulatina e incremental, es decir, cada arquitectura Qi agrega más funcionalidad (ya sean instrucciones nuevas o modos de direccionamiento) a las arquitecturas Qi anteriores.

Incorporar lo que sigue a un gráfico de la estructura en cebolla[18]

3.4.1. Q1

Modos de direccionamiento

1. Inmediato
2. Registro

Instrucciones

1. MOV
2. SUB

[18]

3. DIV
4. ADD
5. MUL

3.4.2. Q2

Modos de direccionamiento

1. Modos de direccionamiento **Q1**
2. Directo

Instrucciones

1. Instrucciones **Q1**

3.4.3. Q3

Modos de direccionamiento

1. Modos de direccionamiento **Q2**

Instrucciones

1. Instrucciones **Q2**
2. CALL
3. RET

3.4.4. Q4

Modos de direccionamiento

1. Modos de direccionamiento **Q3**

Instrucciones

1. Instrucciones **Q3**
2. CMP
3. JMP
4. JE
5. JNE
6. JLE
7. JG
8. JL
9. JGE
10. JLEU

11. JGU
12. JCS
13. JNEG
14. JVS

3.4.5. Q5

Modos de direccionamiento

1. Modos de direccionamiento **Q4**
2. Idirecto
3. RegistroIdirecto

Instrucciones

1. Instrucciones **Q4**

3.4.6. Q6

Modos de direccionamiento

1. Modos de direccionamiento **Q5**

Instrucciones

1. Instrucciones **Q5**
2. AND
3. OR
4. NOT

4. Estado del arte (Susi)

Historia de Orga con respecto a los simuladores que se usaron

Parte II
Simulador QSim

5. Funcionalidad del simulador

La funcionalidad del simulador puede caracterizarse mediante las siguientes partes importantes:

- Chequeo de sintaxis de los programas escritos en el lenguaje Q

- Ensamblado del código fuente de un programa en su correspondiente código máquina
- Cargado en memoria del código máquina
- Ejecución paso a paso de un programa cargado en memoria

5.1. Chequeo de sintaxis

El simulador provee al alumno de un editor de texto en el cual escribirá el programa en un lenguaje Qi, que desea cargar en memoria y ejecutar. Una vez que el usuario haya terminado la escritura, al momento de cargar el programa, el simulador utilizará un parser para detectar errores de sintaxis, tales como la falta de una coma o un corchete, o la presencia de símbolos que no pertenecen al lenguaje (como por ejemplo signos de pregunta y símbolos matemáticos); o bien errores semánticos como la combinación incorrecta de elementos del lenguaje, por ejemplo: modos de direccionamiento mal ubicados. El parser solo revisará lo escrito por el alumno y de acuerdo a las gramática del lenguaje, mostrará alguno de los siguientes estados:

OK Este mensaje se obtiene cuando no hubo ningún error de sintaxis. Si se da este resultado, es posible continuar con el ensamblado y cargado en memoria.

SyntaxError Este mensaje de error se obtiene cuando en alguna línea del programa se detectó algún error de sintaxis o de semántica, como se describió arriba. Cuando ocurre este error se lo acompaña con una descripción lo mas detallada posible para que el alumno detecte donde ocurrió y pueda corregirlo. Un programa con errores no puede ser ensamblado y cargado en memoria.

5.2. Ensamblado

Una vez que el programa es sintácticamente válido es posible traducir el código fuente del programa en código máquina (representado en cadenas binarias). Para esto se respeta un formato de instrucción que indica cómo se codifica cada operación y los operandos.

Mas detalle al respecto de este proceso en la sección de ...[19]

5.3. Cargado en memoria

Una vez ensamblado, la representación binaria (o código máquina) del programa será cargado en memoria a partir de una ubicación (celda de memoria) que el alumno puede elegir. Esto permite visualizar el contenido de la memoria (con el programa cargado) y el estado de los registros de la CPU. La decodificación con desensamblado permite al alumno experimentar otros escenarios y efectos laterares, entre los cuales podemos enumerar:

- Si la ejecución paso a paso excede los límites del programa, pueden tomarse instrucciones de otra rutina y procesarse como una nueva instruccion.

[19] Poner vínculo a donde se ponen ejemplos de Qi

- Si en cambio, se intenta ejecutar el contenido de una celda con datos (y no una instrucción) podrá ocurrir que se encuentre una instrucción invalida (combinacion de modos, codigos, incorrecta) y el alumno verá el mensaje de error pertinente.

Durante la carga del programa en memoria puede producirse un OutOfMemoryError?, que significa que el programa no entra en la ubicación elegida en memoria ya que ocupa más celdas que las que se encuentran disponibles debajo de ella ya que como se mencionó en la sección 3.1, la memoria disponible tiene un tamaño limitado y por este motivo la alocación en memoria del código máquina puede exceder el espacio disponible a partir de la celda inicial anteriormente elegida. Si por el contrario, no se produce este error, el alumno podrá ver el programa cargado en memoria exitosamente.

5.4. Ejecución paso a paso

Se provee la funcionalidad de la ejecución paso a paso ya que se desea que el alumno pueda experimentar y así comprender los pasos del ciclo de ejecución. Además puede ejercitarse situaciones que se denominan "errores conceptuales de programación" Esto es a lo que llamamos Errores conceptuales, entre los cuales es posible mencionar:

- Tomar un dato de un sector de memoria equivocado.
- Que el programa sobrescriba su mismo código máquina.
- Permitir que la ejecución continue una vez terminado el programar cargado en memoria.
-

El paso a paso que provee el simulador consiste en las siguientes estapas pertenecientes al ciclo de instruccion:

1. **Búsqueda de instrucción:** El alumno podrá visualizar el valor que contiene PC (Program counter) donde se encuentra la dirección de la celda en memoria que contiene la próxima instrucción a ejecutar (por ejemplo, en caso de ser la primer instrucción del programa recién cargado, el pc tendrá la dirección de memoria elegida por el alumno para iniciar el cargado del programa en memoria). El simulador, toma de la memoria el código maquina correspondiente a la instrucción que comienza en esa dirección tomada de PC (una instrucción puede ocupar más de una celda de memoria) y los guarda en el registro IR (*Instruction Register*). Será observable también para el alumno el incremento del registro PC, tantas como celdas ocupe la instrucción actual, lo que conceptualmente es, preparar el contexto de ejecución para tomar la siguiente instrucción.
2. **Decodificación:** En la decodificación el Interprete se encarga de desensamblar el código máquina (abreviado en hexadecimal) que ya fue ubicado en el registro IRpara mostrar el código fuente de la instrucción actual con sus respectivos operandos. Si el programa escrito por el alumno es sintacticamente y conceptualmente correcto, este paso le permite comprobar

que la instrucción actual es la que él mismo escribió y no otra, visualizandola en pantalla. En esta etapa se provee también la oportunidad de que el alumno aprecie otros conceptos, tales como los errores conceptuales mencionados antes.

3. **Ejecución** El execute ejecuta los efectos de la instrucción y muestra en pantalla los cambios en el estado de ejecución: memoria, puertos, registros y flags. Dentro de esta misma etapa se lleva a cabo el almacenamiento de resultados que, cuando sea necesario, guardará el valor resultante de la operación descripta por la instrucción en el operando destino. Esto cambiará el valor de una celda de memoria o de un registro y será visto en pantalla por el alumno.

6. Implementación

6.1. Tecnología utilizada (tati)

- **Scala** Elegimos el lenguaje Scala para realizar el simulador ya que, en la materia llamada Objetos III, lo utilizamos un pequeño lapso de tiempo y creímos que era una buena oportunidad para, en vez de elegir un lenguaje que hayamos utilizado más en la carrera como java, profundizar en la utilización de Scala y aprovechar las ventajas que este ofrecia al combinar el manejo de objetos y las características de un lenguaje funcional.

- **Arena** Utilizamos el framework Arena para realizar la interfaz de usuario del simulador porque es un framawork de codigo abierto, que también pudimos utilizar en una de las materias y al poder ser convinado con Scala nos pareció una buena oportunidad de explotar lo que nos ofrecia para que este simulador en su totalidad sea de codigo abierto.

- **Eclipse** Se eligio utilizar el entorno de programación Eclipse ya que ambas trabajamos en diferentes sistemas operativos para los cuales Eclipse es funcional y puede tener los mismos pluggins que hace que pueda sopotar proyectos MVN y Scala, además de que, al ser el entorno de programación que más hemos usado nos sentiamos cómodas con esa elección.

- **Git** Elegimos git como repositorio externo para sincronizar nuestro codigo ya que promueve el codigo abierto (ACA QUIERO PONER LO DE QUE ESTA AHI ARRIBA Y NO LE CEDEMOS DERECHOS NI NADA A NADIE PERO NO ME SALE!!!)

6.2. (arq OO)

Como se observa en la figura 11 tiente toda la responsabilidad en la ejecución de operaciones matematicas y logicas, además del analisis sobre los flags luego de cada operación.

Como se observa en la figura 2 el Bus de entrada y salida tiene la responsabilidad de derivar según donde corresponda (Memoria o Puertos) la modificación de una celda o el leer un dato. Para conoce a una instancia de la clase Memoria y a otra de la clase CeldasPuertos. Ambas clases conocen muchas intancias de

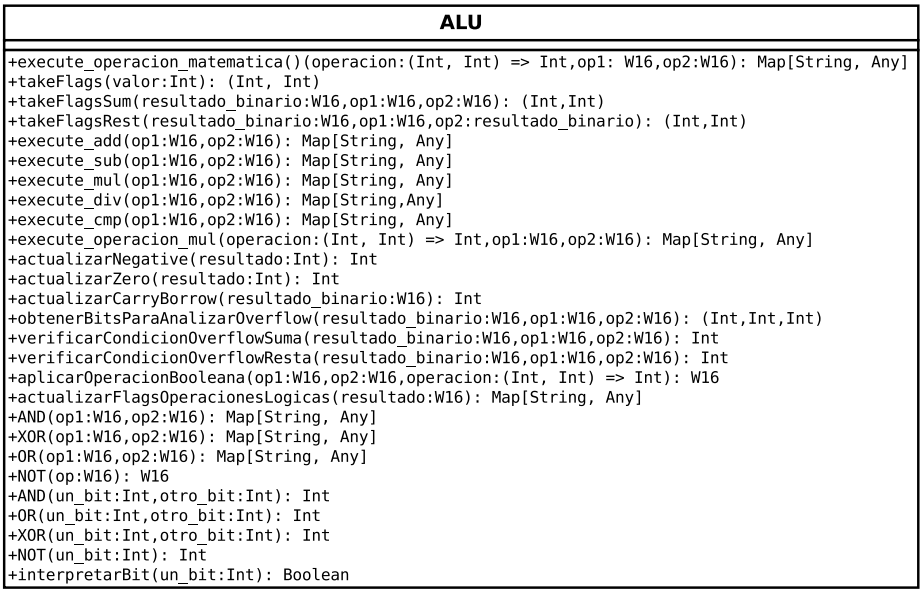

Figura 1: Diagrama de clase de la Unidad Aritmético-Lógica

la clase Celda, y cada Celda a su vez conoce un dato: una instancia del W16, que representa a los datos guardados en memoria o en los puertos.

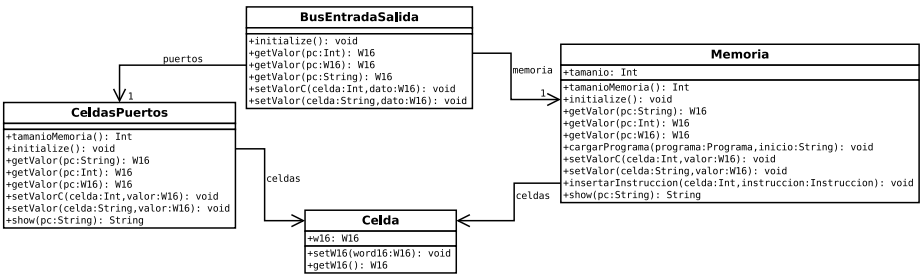

Figura 2: Diagrama de clase del la $_{M}emoria_{C}eldasPuertos$

Como se observa en la figura 11
Como se observa en la figura 11
Como se observa en la figura 11
Como se observa en la figura 11
Como se observa en la figura 11
Como se observa en la figura 11
Como se observa en la figura 11
Como se observa en la figura 11
Como se observa en la figura 11

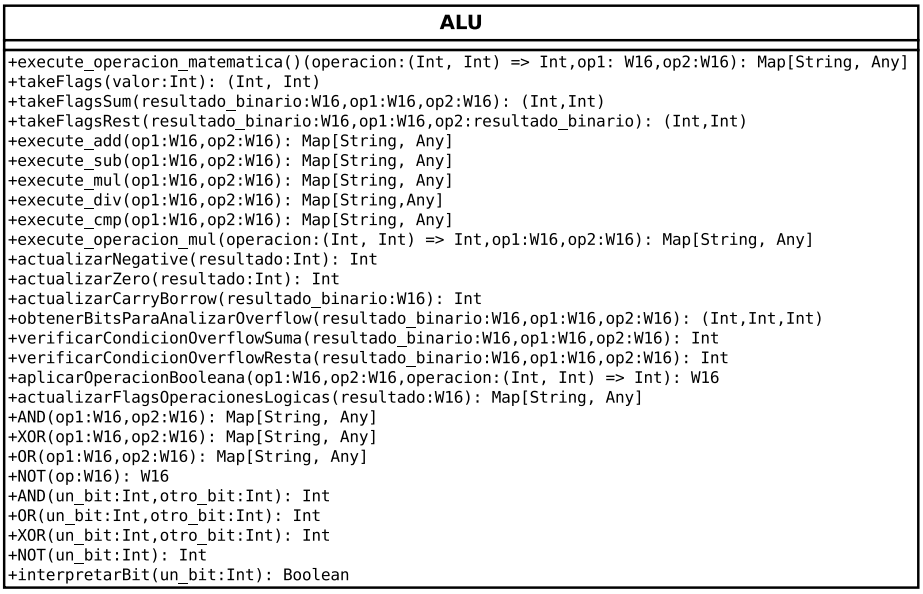

Figura 3: Diagrama de clase de la Unidad Aritmético-Lógica

Parte III
Evaluación del desarrollo

7. Dificultades encontradas

7.1. Dificultades presentadas por el dominio

Las dificultades del dominio estuvieron relacionadas a la comprensión no solamente del modelo de arquitectura Q si no también a su propósito didáctico, ya que el objetivo del simulador no es solamente la simulación de la arquitectura y la ejecución de programas, sino también el proveer a los alumnos la capacidad de ejercitar situaciones conceptualmente erróneas, por lo que se requería una comprensión didáctica del problema más allá de la especificación de las arquitecturas Q.

7.2. Dificultades de diseño

En primera instancia se opto por implementar un modelo de objetos que utilizaba un objeto de la clase **Programa** a lo largo de toda la ejecución. Procesaba las instrucciones no leyendo de la matriz memoria, si no, pidiendo la siguiente instrucción al objeto instancia de la clase Programa, sobreviviendo así las distintas etapas una vez que fue creado y evitando la creación de un objeto cuya responsabilidad sea interpretar el código máquina alojado en la memoria, evitando la nueva creaccion de instancias de la clase **Instruccion** y haciendo mucho más sencillo al modelo ya que la memoria era sólo una clase que contenia datos que se reflejaban en pantalla y no se extraian datos de celdas en ningun momento. Luego entendimos que un programa no sólo podía modificar su entorno al ser

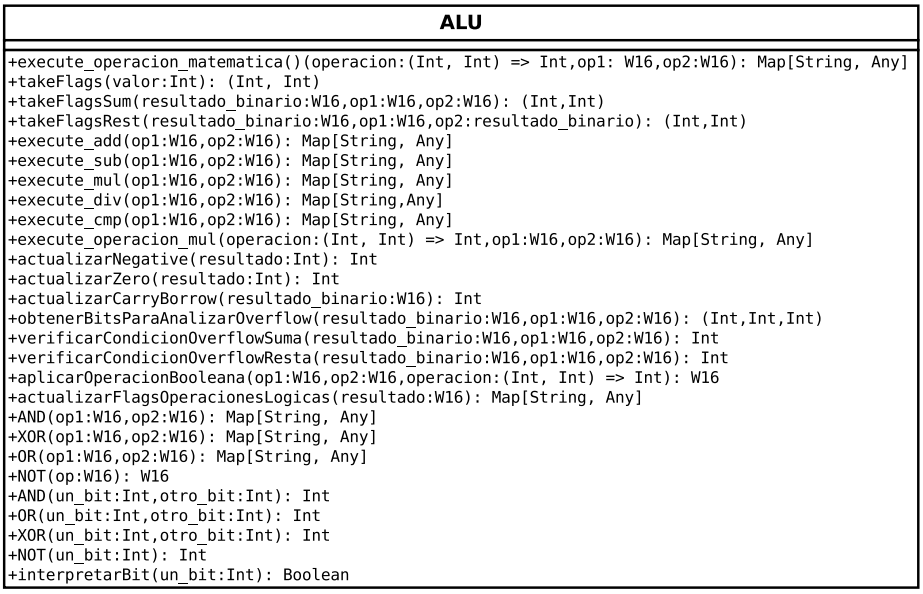

Figura 4: Diagrama de clase de la Unidad Aritmético-Lógica

ejecutado (otras celdas de memoria que no ocupen su código maquina, celdas de puertos, registros, etc) si no que también podría sobreescribir su código maquina (ya sea con ese propósito o sólo por un error conceptual), o bien, que el alumno debía tener la posibilidad de seguir ejecutando más allá del código máquina alojado en memoria o más, inevitablemente se necesitó corregir gran parte del modelo agregando una clase denominada **Intérprete**, cuya responsabilidad es interpretar la siguiente instrucción alojada en memoria para que luego sea ejecutada, otorgando más responsabilidad a la clase **Memoria** y descartando el objeto instancia de **Programa** una vez que éste es cargado en memoria con éxito.

Tuvieron que ser solicitadas además extensiones al equipo de desarrolladores de Arena para poder realizar la interfaz con dicho framework.

- FileSelector para que los alumnos puedan cargar el archivo .Qsim en el cual se encuentra su programa
- CodeEditor (O actualmente llamado KeywordTextArea). Es el espacio donde se visualiza el programa QUARQ que realizaron los alumnos una vez que esta cargado.
- Bindings contra el background de componentes y celdas de una tabla, para permitir la visualizacion de cambios de colores en la memoria luego de las etapas de ejecución y de realizar algun cambio en la misma.
- TexBox multiLine (TextArea) con scroll para ser utilizado como consola de devolucion.
- Icono para la aplicacion, solo por cuestiones estéticas.

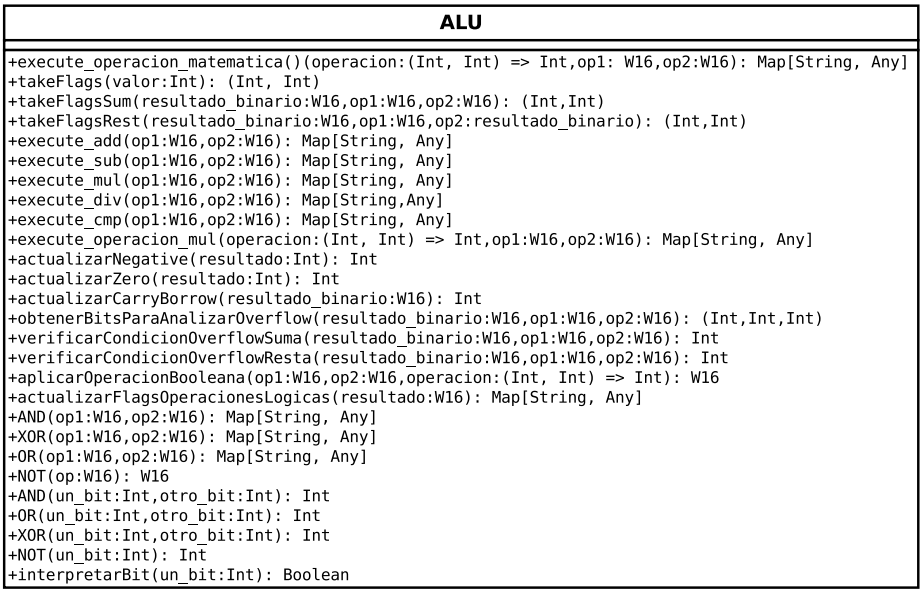

Figura 5: Diagrama de clase de la Unidad Aritmético-Lógica

8. Casos de prueba

8.1. Chequeo de sintaxis en la distintas Qi

Para chequear la sintaxis de cada **Arquitectura Q**, primero escribimos un programa Qi en un archivo con extension .qsim luego este archivo lo recibe el objeto Parser que se encargara de chequear la sintaxis. Para eso realizamos dos casos de prueba por cada Qi:

1. **Chequear programa Qi valido**

   Agarramos el programa Qi valido y se lo pasamos al Parser, cuando termina de chequear devuelve un resultado, como es de esperar devuelte un objeto programa con la lista de instrucciones. Tomamos ese resultado y como ultimo paso comparamos si el programa resultado es igual al programa esperado.

   (esto va en nota al pie)programa Qi valido = Es un programa en cualquier arquitectura Q escrito sintaticamente correcto.

2. **Chequear programa Qi es invalido**

   En este caso escribimos un programa Qi con errores de sintaxis para ver como reacciona el parser.

   Agarramos el programa Qi invalido y se lo pasamos al Parser, cuando termina de chequear devuelve un error, como es de esperar devuelve una Exception porque dentro del archivo del programa hay una linea que tiene errores de sintaxis, el mensaje de la excepcion te muestra el numero de linea y la propia linea para que veas en donde te equivocaste. Capturamos la Excepcion y tomamos como resultado el mensaje y como ultimo paso comparamos el mensaje resultado con el mensaje esperado.

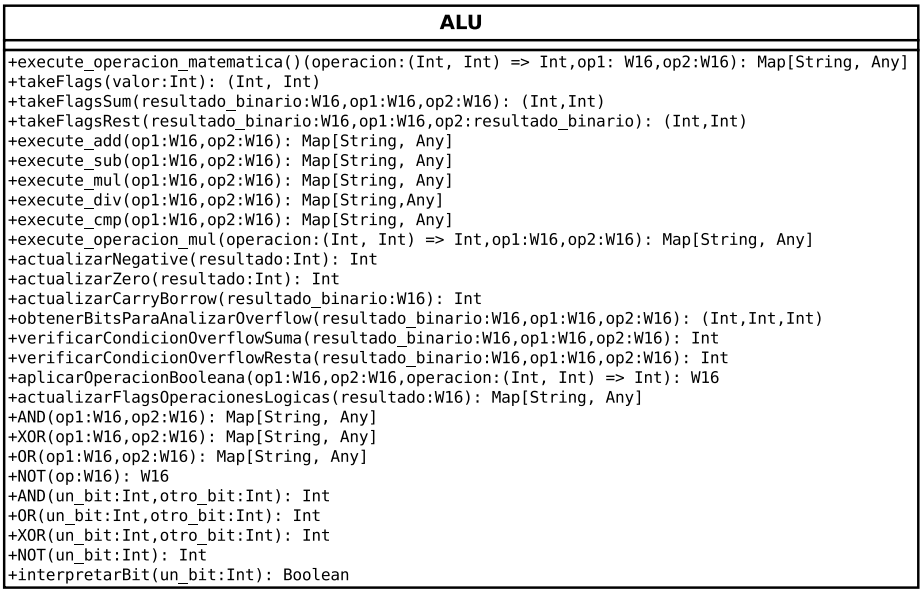

Figura 6: Diagrama de clase de la Unidad Aritmético-Lógica

9. Ejemplos de uso

Referencias

[1] Williams Stallings, *Computer Organization and Architecture*, octava edición, Editorial Prentice Hall, 2010.

[2] A. Tanenbaum, *Organización de Computadoras*, cuarta edición, Editorial Pearson.

[3] Hennessy, Patterson. *Arquitectura de Computadores - Un enfoque cuantitativo*, primera edición, Editorial Mc Graw Hill.

[4] Sitio oficial de la materia Organización de Computadoras: `http:\orga.blog.unq.edu.ar` (2013)

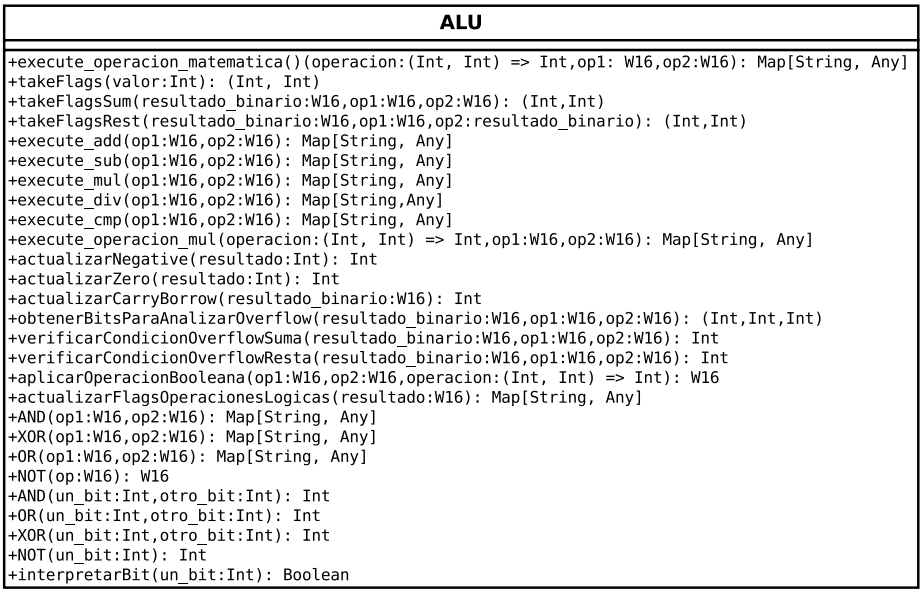

Figura 7: Diagrama de clase de la Unidad Aritmético-Lógica

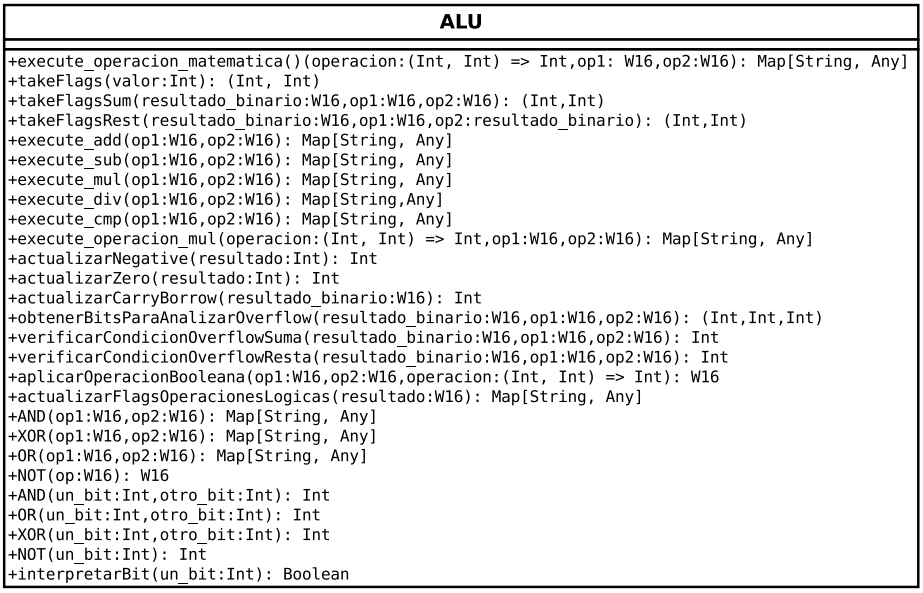

Figura 8: Diagrama de clase de la Unidad Aritmético-Lógica

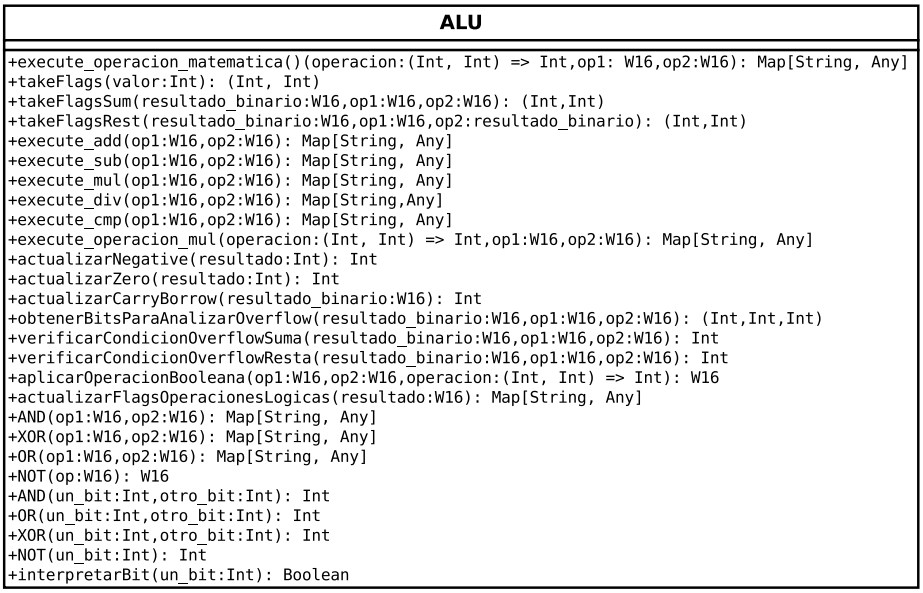

Figura 9: Diagrama de clase de la Unidad Aritmético-Lógica

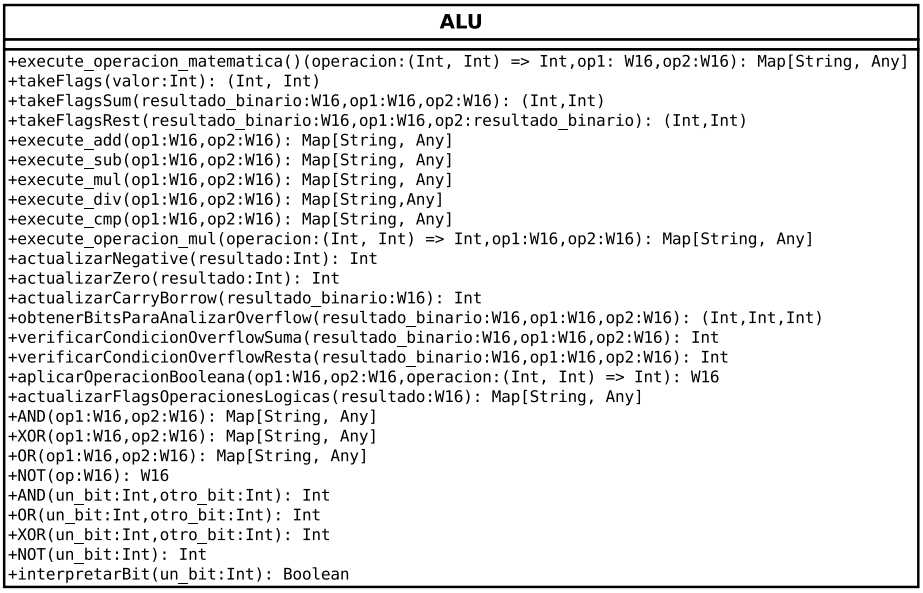

Figura 10: Diagrama de clase de la Unidad Aritmético-Lógica

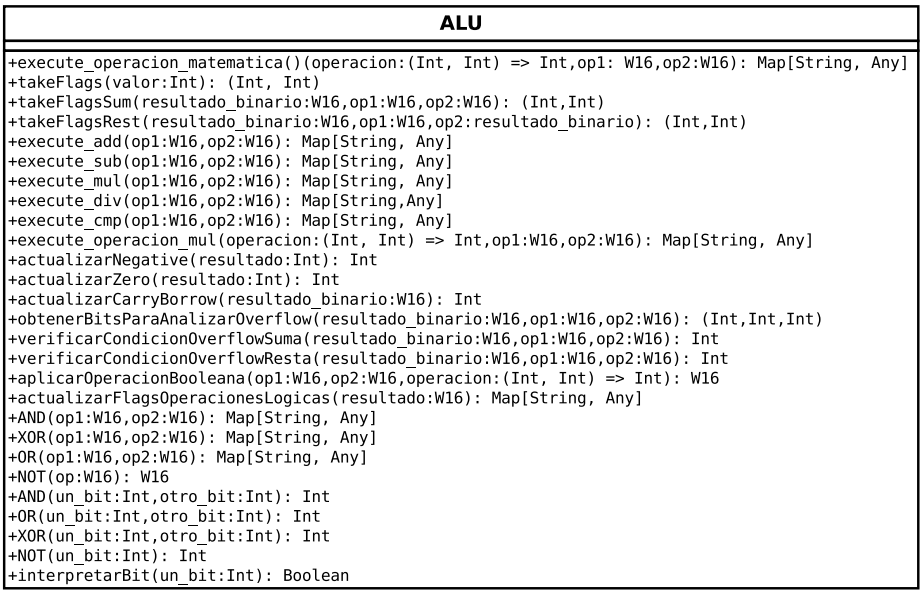

Figura 11: Diagrama de clase de la Unidad Aritmético-Lógica