FACULTAD 50 AÑOS DE INGENIERÍA UNaM
JIDeTEV 2024 ISSN 2591-4219
XIV Jornadas de Investigación, Desarrollo Tecnológico, Extensión, Vinculación y Muestra de la Producción

Arquitectura del Conjunto de Instrucciones Pedagógico para el Inicio en la Carrera de Ingeniería en Computación

Thiago A.Cruz [a], Hernán F. Kisiel [a], Roberto E. Carballo [a,b,*]

[a] *Universidad Nacional de Misiones, Facultad de Ingeniería, Oberá, Misiones, Argentina.*
[b] *GIDE, FI-UNaM, Oberá, Misiones, Argentina.*

cruzthiagoa@gmail.com, hernankisiel@gmail.com, robertocarballo@fio.unam.edu.ar.

Resumen

Se propone una arquitectura del conjunto de instrucciones (ISA – *Instruction Set Architecture*) que tiene como objetivo la enseñanza de los fundamentos del funcionamiento de un procesador, enfocado a estudiantes de ingeniería en computación que inician sus estudios en la carrera. Se presentan las motivaciones para la propuesta, los objetivos para la elección de las instrucciones, su formato y modos de direccionamiento. Finalmente se presentan 2 ejemplos de cómo resulta la programación de esta computadora hipotética.

Palabras Clave *– Arquitectura del Conjunto de Instrucciones Pedagógico, Microarquitectura, Procesador de un Solo Ciclo. Fundamentos de Informática.*

Introducción

Existen varios enfoques para introducir a los estudiantes al funcionamiento de las computadoras modernas en las carreras vinculadas a las ciencias de la computación, pudiéndose agrupar estos en dos clasificaciones, enfoques de abajo hacia arriba o de arriba hacia abajo [1-4].

Los enfoques de abajo hacia arriba consisten en comenzar por enseñar los componentes de la computadora pero desde sus partes constitutivas, por lo que se suele comenzar por presentar en primer lugar el transistor MOSFET, el cual es la base para la construcción de circuitos CMOS y luego con estos como se construyen las compuertas lógicas, para seguir con el armado de circuitos combinacionales, circuitos aritméticos y memorias. De esta forma se van construyendo y agrupando los distintos niveles de abstracción que conforman las bases de las computadoras modernas, llegando hasta la microarquitectura de un procesador y su Arquitectura del Conjunto de Instrucciones (ISA – *Instruction Set Architecture*), e inclusive continuar con la enseñanza de lenguajes de programación *assembly* y de alto nivel. El objetivo del enfoque de abajo hacia arriba es que el estudiante construya su conocimiento sobre lo que ya sabe, aprendió o va aprendiendo en el transcurso del curso, minimizando los vacíos que puedan haber en el funcionamiento de cada componente y sus interacciones [5].

Los distintos niveles de abstracción que se tienen desde el problema que se quiere resolver usando una computadora (o redes de computadoras), hasta los electrones que fluyen por los circuitos transistorizados de la mismas, se agrupan en lo que se denomina la jerarquía de transformación [3], presentada en la Fig. 1.

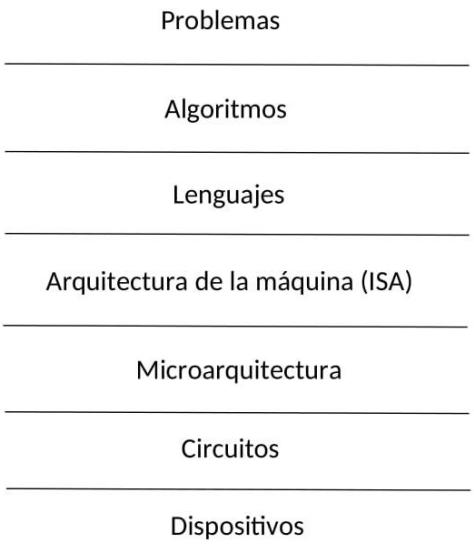

Fig. 1 Jerarquía de Transformación [3].

A diferencia de los enfoques de abajo hacia arriba, los enfoques de arriba hacia abajo se basan en enseñar primero a programar en un lenguaje de alto nivel, con lo cual suele variar mucho dependiendo del curso que lenguaje se enseña. El argumento en contra del por qué no es bueno este tipo de enfoques es que el estudiante termina memorizando y tratando de aplicar patrones que aprendió en ejercicios previos a los problemas futuros [6], sin comprender el funcionamiento básico de un procesador y la memoria, lo cual lo lleva a fracasar en la resolución de nuevos problemas. El argumento a favor es que el tiempo para aprender a programar podría ser más reducido que con un enfoque de abajo hacia arriba, sacrificando un conocimiento profundo sobre los componentes de la computadora y su funcionamiento.

Por lo general en los enfoques de abajo hacia arriba se suele utilizar un modelo de máquina (descrito por su ISA) que permite al estudiante entender cómo funciona una computadora, como es la interacción memoria-procesador, la programación a bajo nivel, ya sea en código máquina o el lenguaje *assembly* de una máquina en particular. Los modelos utilizados generalmente son en base a arquitecturas RISC (*Reduced Instruction Set Computer*, Arquitectura del Conjunto de Instrucciones Reducida), ya que otorgan pocas instrucciones con las que el estudiante debe familiarizarse. Estos modelos suelen estar acompañados de herramientas para simular la ejecución del código, así es posible escribir algoritmos y probar su funcionamiento en un entorno de simulación [7].

Dentro de los posibles modelos de máquinas más simples para aprender el funcionamiento de las computadoras, se encuentra una propuesta del profesor Yale Patt, denominada LC-3 (*Little Computer* – 3, Pequeña Computadora 3), para la cual definió una ISA pedagógica basada en arquitecturas RISC, con 14 instrucciones y 5 modos de direccionamiento.

Si bien la ISA propuesta por Patt Y. es simple y permite a los estudiantes introducirse en forma más amigable a conceptos como, código máquina, lenguaje *assembly*, microarquitectura de un procesador, en comparación a si lo tuvieran que hacer con MIPS, ARM o RISC-V (enfoques utilizados en libros de los autores Harris S. y Harris D. [8-10]), pero requiere preestablecer las siguientes características de un procesador, las cuales no son tan intuitivas para un estudiante que recién comienza:

1. Es necesario un banco de registros.
2. Se debe aprender las características de instrucciones de cargar (*Load*) y guardar (*Store*) en 3 diferentes modos de direccionamiento: relativo al contador de programa, indirecto y base + *offset*.
3. Se tienen varios campos dentro de los formatos de las instrucciones, ya que se cuenta con el *opcode*, las direcciones destino y fuente dentro del banco de registros, la dirección base en un banco de registros, el bit de dirección (bit 5 en instrucciones de procesamiento), los bits inmediatos (constantes), los bits NZP en instrucciones de salto condicional y los distintos *offsets*.
4. Se deben aprender otras instrucciones que facilitan o permiten implementar funcionalidades que pueden considerarse "más avanzadas" que la implementación de algoritmos básicos, como el manejo de interrupciones (instrucción RTI), llamado a subrutinas (instrucciones JSR, JSRR, RET) y llamadas al sistema operativo (instrucción TRAP).

El hecho de tener que explicar por qué se utiliza un banco de registros presenta desafíos al enseñar estos conceptos a los estudiantes. Esto se debe a que es necesario introducir nociones de jerarquía de memoria, cachés y pipelines, que son las estructuras de hardware que permiten aprovechar las ventajas de incorporar bancos de registros en las máquinas modernas y, en conjunto, son las principales responsables de mejorar el rendimiento.

Contar con un modelo de máquina más simple, que opera directamente desde la memoria y con formatos de instrucción más reducidos que los de la LC-3, permitirían lograr que en los tiempos de cursado que tiene la materia Fundamentos de Informática de la FIO (60 hs en un cuatrimestre), poder mejorar el nivel de comprensión de los estudiantes en el funcionamiento de una computadora. Con esta motivación en este trabajo se propone un modelo de máquina aún más simple que la LC-3, la cual se basa en una máquina de pila (*stack machine*) que solo requiere un acumulador, 5 instrucciones para operarla, cuenta con un formato simple para cada una de estas, y solo dos modos de direccionamiento.

El modelo propuesto presenta un inconveniente: la imposibilidad de direccionar toda la memoria con una sola instrucción. Sin embargo, este problema se resuelve en la LC-3, lo que permitirá al estudiante abordarlo en la materia de Arquitectura de Computadoras.

La ISA presentada en este trabajo está diseñada para ser el punto culminante del curso de Fundamentos de Informática, impartido en el primer año de la carrera. El objetivo es que en el segundo año, la materia de Arquitectura de Computadoras pueda retomar y expandir naturalmente los conocimientos adquiridos, utilizando la LC-3 como una continuación lógica de lo aprendido previamente.

Este trabajo se encuentra organizado de la siguiente forma: 2) ISA propuesto y su Microaquitectura, 3) Ejemplos de programa, 4) Conclusiones.

2 ISA propuesto y su Microarquitectura

2.1.*Conjunto de instrucciones y sus formatos*

En primera instancia del desarrollo de la ISA se planificó contar con instrucciones de procesamiento, memoria y control de flujo de programa, ya que estos tres tipos de instrucciones

son los que se encuentran en la LC-3. La diferencia principal con las instrucciones de la propuesta de Patt Y. es que las instrucciones de memoria mueven datos entre memoria y el acumulador, y no entre memoria y un banco de registros. Esto facilita la operación entre memoria y procesamiento, ya que no se tienen que cargar registros antes de comenzar a operar, pero sacrificando simplicidad en el programa, ya que ahora se requieren más instrucciones de memoria para llevar y sacar los datos del acumulador.

Para mantener compatibilidad con las instrucciones de procesamiento de la LC-3, se plantea mantener las mismas instrucciones, ADD (suma aritmética), AND y NOT, estando la diferencia en que, debido a la estructura de acumulador con la que trabaja la ALU (*Arithmetic Logic Unit*, Unidad Aritmético Lógica), se tiene un solo operando. En la Fig. 2 se presenta la conexión simplificada de la ALU con el acumulador.

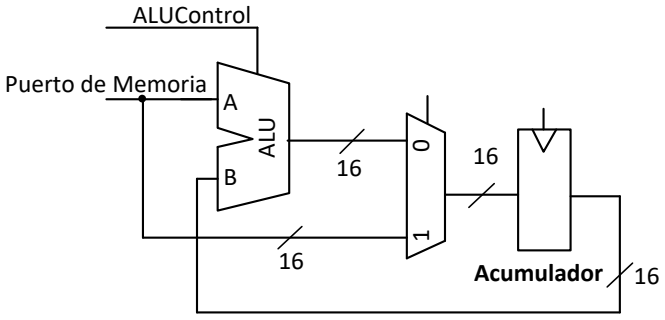

Fig. 2: Conexión de la ALU con la Memoria y el Acumulador.

El formato de las instrucciones de procesamiento se presenta en la Fig. 3, donde es posible observar el campo de *opcode* (13:15) (*operation code*, código de operación), el bit 12 que hace de bit de dirección (lo que se denomina como *steering bit*) y define el modo de direccionamiento en este caso, pudiéndose utilizar el modo relativo al contador de progama (bit 12 = 0) o inmediato (bit 12 = 1).

	15 14 13	12	11 10 9 8 7 6 5 4 3 2 1 0
ADD$^+$	0 0 0	0	PCoffset12
ADD$^+$	0 0 0	1	imm12
AND$^+$	0 0 1	0	PCoffset12
AND$^+$	0 0 1	1	imm12
NOTA$^+$	0 1 0	0	PCoffset12
NOTB$^+$	0 1 0	1	0 0 0 0 0 0 0 0 0 0 0 0

Fig. 3: Formato de las instrucciones de procesamiento.

La instrucción NOT tiene dos modos de direccionamiento con bit 12 definiéndolos, el relativo al contador de programa (NOTA) y el modo acumulador (NOTB), haciéndose en este último NOT bit a bit sobre el contenido del acumulador directamente.

El multiplexor en la Fig. 2 es para utilizarlo con instrucciones de memoria, ya que se propone poder cargar el Acumulador directamente con lo que provenga de la memoria. Para esto se utilizaría la instrucción LD, que viene del termino *load* (cargar). Se optó este nombre para mantener la similitud con la instrucción LD de la LC-3, la cual a diferencia de esta ISA propuesta permite cargar un registro del banco de registros en la LC-3.

Completando las instrucciones de memoria tenemos ST, proviene de *store* (guardar), el cual transfiere el contenido del acumulador a la memoria en modo relativo al contador de programa.

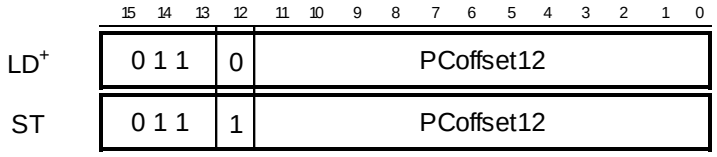

Fig. 4: Formato de las instrucciones de memoria.

Notar que el bit 12 en el formato presentado en la Fig. 4 cambia la dirección de hacia donde irían los datos, con bit 12 = 0 se lleva de memoria a acumulador, y con bit 12 = 1 de acumulador a memoria.

En las figuras Fig. 3 y Fig. 4 el superíndice con un signo + indica que esas instrucciones modifican los bits N, Z, P en el códigos de condiciones (CC), que se utiliza para comparar con los bits NZP de la instrucción de control de flujo de programa BR (la abreviación proviene de *branch*, ramificar), la cual se presenta su formato en la Fig. 5.

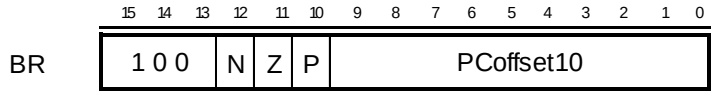

Fig. 5: Formato de la instrucción de control de flujo de programa.

La instrucción BR permite cambiar la dirección del PC (contador de programa) si existe una coincidencia en algunos de los bits NZP con los bits NZP del código de condiciones. En caso de coincidir se accede a la siguiente instrucción en ejecución dada por la dirección PC+1+PCOffset10, en caso contrario a la correspondiente a PC+1. El PC apunta a un puerto de dirección de memoria, lo cual al leer esta se obtiene una instrucción. Leer otros puertos de dirección de memoria mediante las instrucciones LD o ST permite obtener o almacenar datos en la memoria. Las conexiones relevantes se detallan en la siguiente sección, donde se presenta la microarquitectura del procesador que utiliza esta ISA.

2.2. *Microarquitectura*

Para la microarquitectura del procesador se considera que toda la instrucción se ejecuta en un solo ciclo de reloj (lo que se conoce como procesador de un solo ciclo), pudiendo mapear cada instrucción de la ISA en forma directa al hardware que las ejecuta, obteniéndose la representación de la Fig. 6.

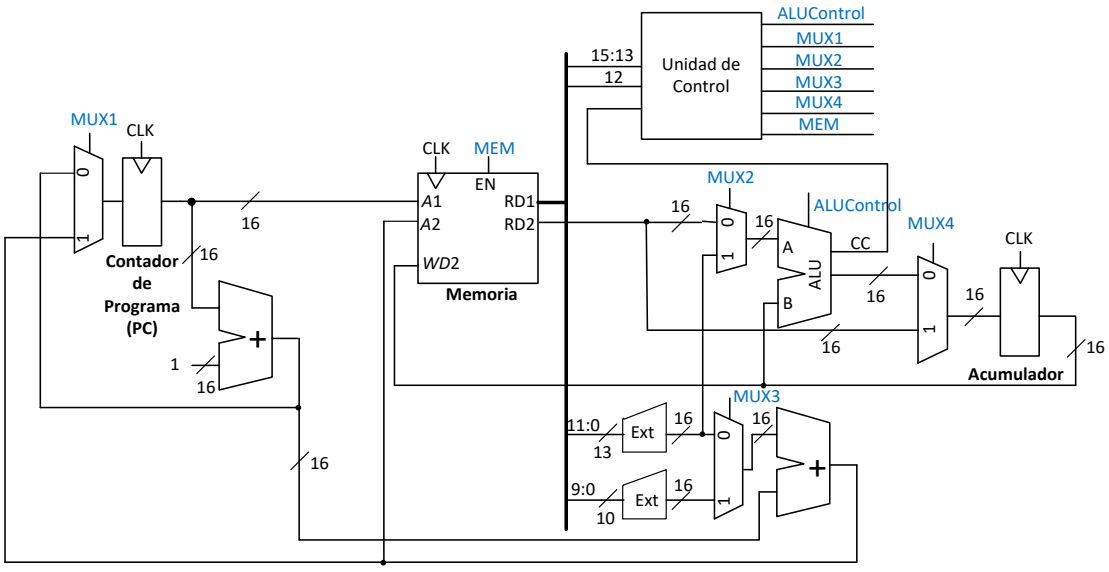

Fig. 6: Microarquitectura del procesador propuesto con implementación en un solo ciclo de reloj.

Para comparar la simplicidad de la microarquitectura de la ISA propuesta con la de la LC-3, que también se implementa en un solo ciclo de reloj (siendo en realidad un subconjunto de la ISA de la LC-3 que se aborda en la asignatura de Fundamentos de Informática), se presentan el camino de datos y la unidad de control de la LC-3 con las siguientes instrucciones: ADD, AND, NOT, LD, ST, LDI, STI, LDR, STR, LEA, BR y JMP.

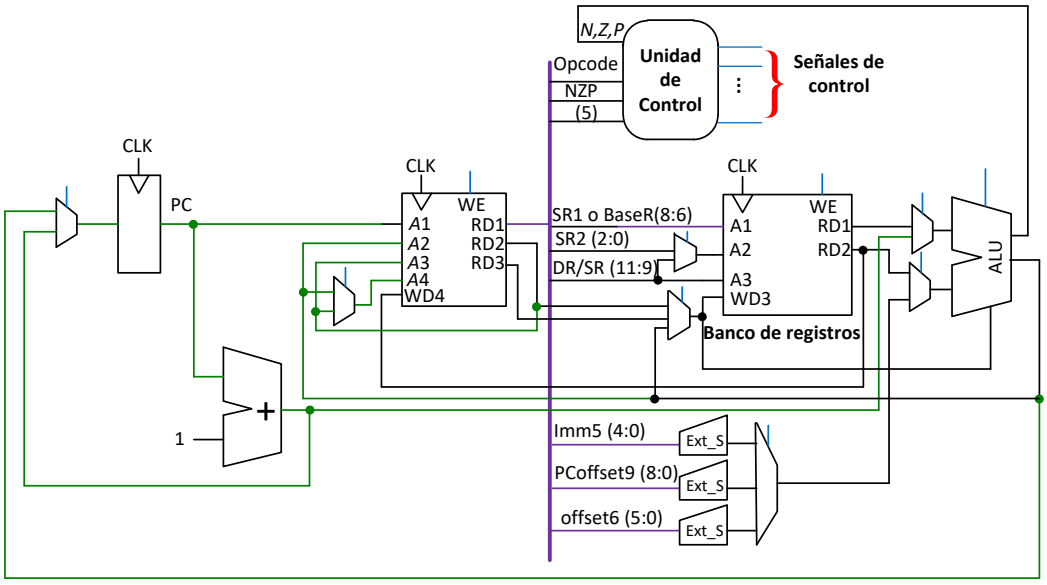

Fig. 7: Microarquitectura de un subconjunto de la LC-3 con implementación en un solo ciclo de reloj, compuesto por: ADD, AND, NOT, LD, ST, LDI, STI, LDR, STR, LEA, BR, JMP.

1. Ejemplos de programa

Para mostrar cómo serían los programas del procesador propuesto se presentan dos ejemplos simples, uno para realizar el cómputo de una OR entre dos números y otro para el cómputo de una multiplicación aritmética entre dos números.

Se considera que el ensamblador reemplaza etiquetas por direcciones de memoria cuando se necesita, facilitando la escritura del programa en lenguaje *assembly*. Esto es una tarea que se tendrá que desarrollar como trabajo a futuro, de forma de brindarles una herramienta de ensamblador a los estudiantes, junto con un simulador del código en *assembly* o código máquina directamente. También se considera que las pseudo-instrucciones son las mismas que las que se usan para programar la LC-3, por lo tanto se cuenta con .ORIG para definir el origen del programa en la memoria, .END para el final, . FILL para asignar valores a la memoria y .BLKW para reservar un espacio de memoria.

```
    .ORIG 0x0000

        NOTA X; X es una variable a la cual se le va a realizar una operación OR con
otra variable Y, por lo que se requiere hacer (X'*Y')', esta primer instrucción hace NOT
bit a bit de X.

        ST X; se guarda X negada en la memoria en la misma ubicación de X.

        NOTA Y; se niega la variable Y, queda Y negada en el acumulador.

        AND X; se hace AND entre X negada e Y negada.

        NOTB; se niega el resultado del acumulador, ya se tendría el resultado OR entre
X e Y.

        ST R; se guarda en otra ubicación de memoria el resultado R, también se podría
haber remplazado en lugar de X o Y.

    X   .FILL #10

     Y  .FILL #18

     R  .BLKW 1

        .END
```

A continuación se desarrolla el programa para la multiplicación aritmética entre dos números, los cuales se denominarán X e Y, y para los cuales en este caso no se sobrescriben sus valores.

```
            .ORIG 0x3000 ; se define el origen del programa en la dirección 0x3000

        LD Y; se carga la variable Y, que se transferirá a una variable temporal CONT, la
cual hará de contador y se irá decrementando.

        ST CONT; se mueve Y a CONT.

ACC     LD X; se carga el acumulador con X;
```

```
        ADD R; se suma a X el resultado previo R;
        ST R; se guarda el resultado R.
        LD CONT; carga contador en el acumulador
        ADD #-1; se decrementa 1 contador.
        ST CONT
        BRp ACC; sí el valor de CONT es positivo sigue acumulando X, sino termina, al finalizar el resultado de la multiplicación estaría en R.
X       .FILL #5
Y       .FILL #3
CONT .FILL#0
R       .FILL #0
        .END
```

Con estos dos ejemplos se observa cómo es posible programar algoritmos utilizando una máquina de pila, la cual contiene los conceptos básicos de código máquina, lenguaje *assembly*, campos de una instrucción, tipos de instrucciones y modos de direccionamiento.

2. Conclusiones

Se ha presentado una arquitectura del conjunto de instrucciones (ISA) de un procesador pedagógico, para ser utilizado en el curso de Fundamentos de Informática de la Carrera de Ingeniería en Computación de la FIO. La ISA contiene 5 instrucciones, 3 de procesamiento, 2 de memoria y 1 de salto condicional, con las que es posible formular cualquier algoritmo simple de forma práctica, esperando que facilite la comprensión de las características de una ISA a estudiantes que recién se inician en la carrera.

Se contrastó la microaquitectura del procesador propuesto con el que surge del subconjunto de la LC-3 con la que se enseñó hasta este año en la materia Fundamentos de Informática. Si bien resulta significativamente más reducida la cantidad de conexiones, se puede apreciar que características como el contador de programa, memoria e interfaz con extensión de signo con la ALU siguen manteniéndose en el camino de datos, por lo que se puede considerar un paso previo en la comprensión de la microarquitectura de ISAs más complejos.

Como trabajos futuros se pretende desarrollar una herramienta de programación y simulación, que permita evaluar el código desarrollado para este procesador. En esta herramienta se debería incluir características del manejo de puertos entrada salida para que puedan intercambiar datos en forma dinámica con la memoria, herramientas de depuración de código y códigos de errores de ensamblado.

3. Referencias

[1] M. Guzdial and B. Ericson, *Introduction to computing and programming in python*: Pearson New York, NY, 2016.

[2] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*: Elsevier, 2011.

[3] S. Patel and Y. Patt, *Introduction to Computing Systems: from bits & gates to C & beyond*: McGraw-Hill Professional, 2019.

[4] P. Jamieson, D. Davis, and B. Spangler, "The Mythical Creature Approach-A Simulation Alternative to Building Computer Architectures," in *FECS*, 2010, pp. 23-28.

[5] Y. N. Patt and K. J. Compton, "Introduction To Computing The Correct (Bottom Up) Approach," in *1997 Annual Conference*, 1997, pp. 2.265. 1-2.265. 15.

[6] Y. Patt. *Yale N. Patt, The future of "Computer *" (Are we in serious trouble?)* Available: https://www.youtube.com/watch?v=4McGiByqDII&t=523s

[7] CPUlator. Available: https://cpulator.01xz.net/

[8] S. Harris and D. Harris, *Digital design and computer architecture: arm edition*: Morgan Kaufmann, 2015.

[9] S. Harris and D. Harris, *Digital Design and Computer Architecture, RISC-V Edition*: Morgan Kaufmann, 2021.

[10] S. Harris and D. Harris, *Digital design and computer architecture*: Morgan Kaufmann, 2015.