Máquina de Von Neumann

Lenin G. Falconí

2024-12-05

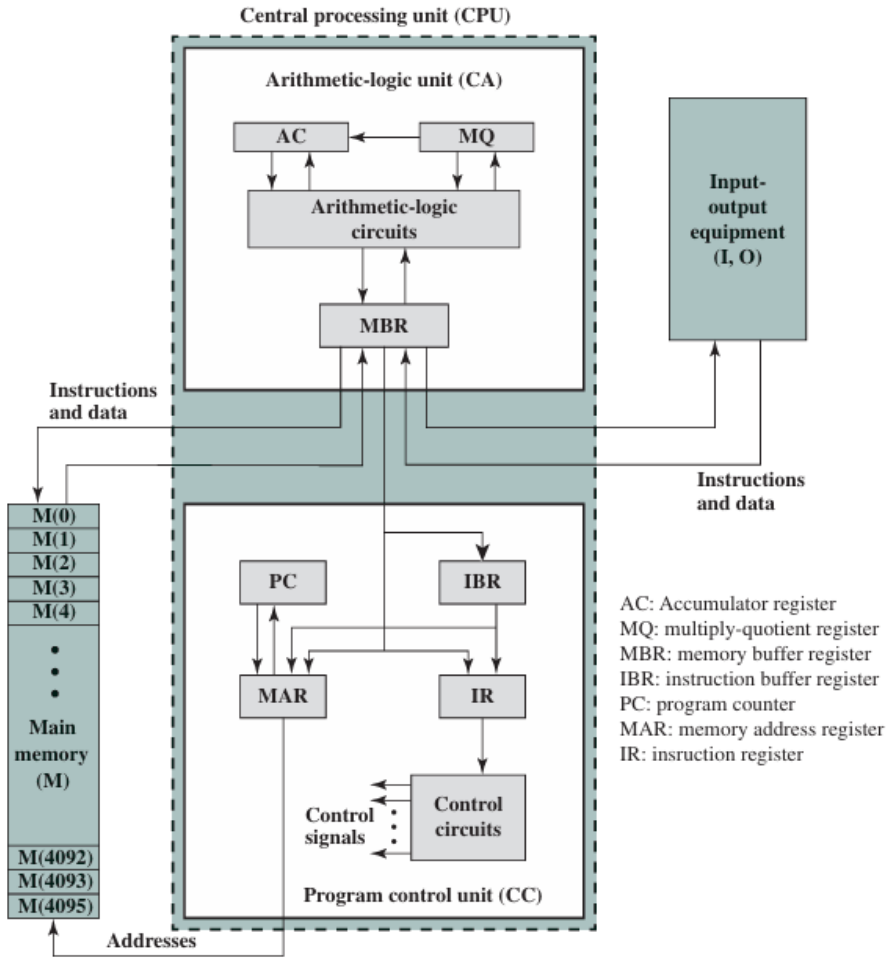
Outline

Conceptos de Diseño I

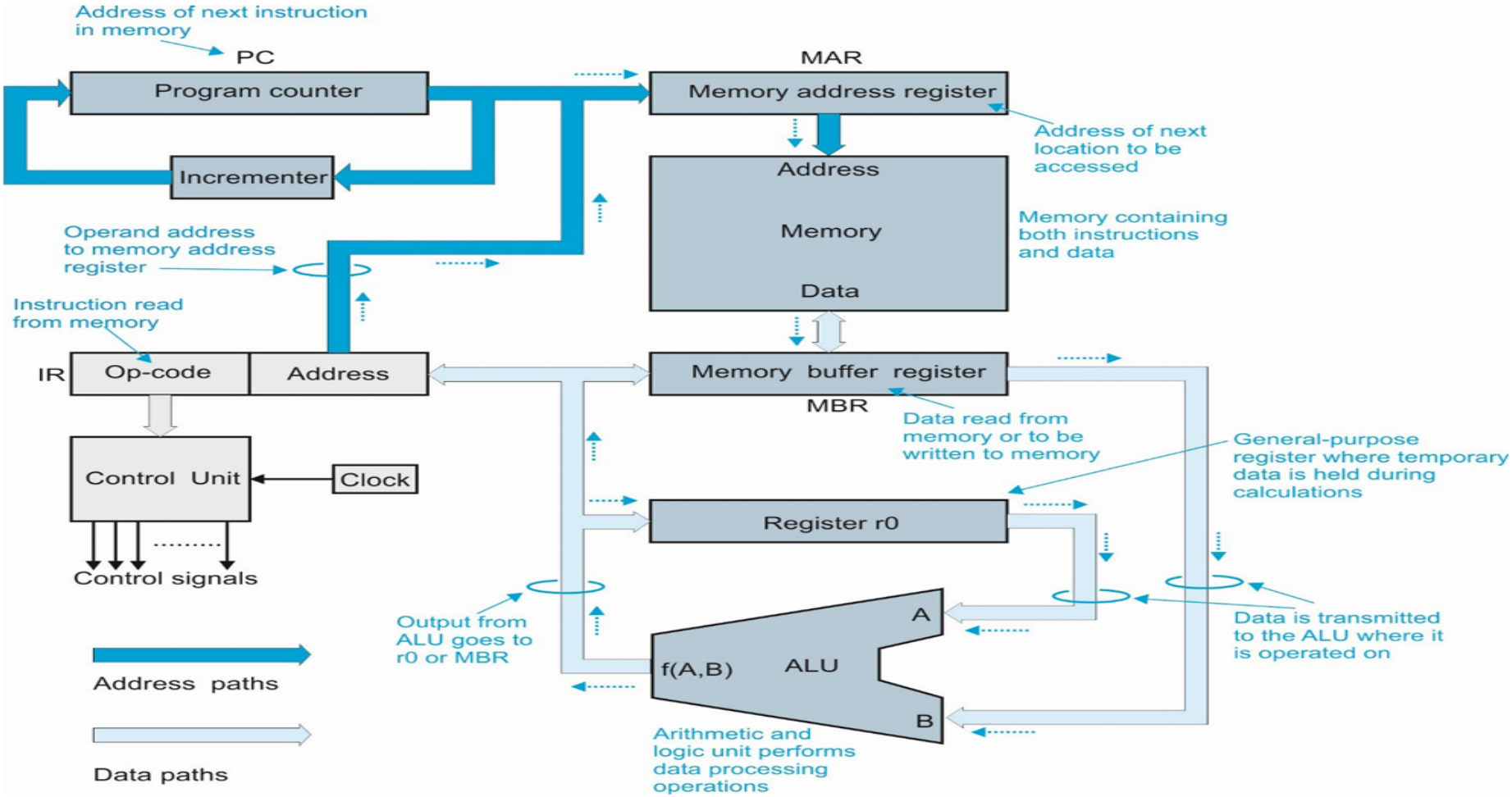
- Los datos y las instrucciones se almacenan en una memoria principal de lectura/escritura
- Los contenidos de la memoria son accesibles por medio de su ubicación y son independientes del contenido o tipo de dato almacenado
- La ejecución ocurre de manera secuencial (aunque puede ser modificado explícitamente e.g. una instrucción de salto)
- El mismo hardware puede desarrollar diferentes funciones sobre los datos dependiendo de las señales de control aplicadas
- El programa, conjunto de códigos de instrucciones y datos, indica las señales de control requeridas.
- La programación usa el mismo hardware pero diferentes códigos para diferentes propósitos.

Computador IAS(Institute for Advanced Study) I

- La idea de diseño: *stored-program concept*
- Idea de diseño de Von Neuman
- Consiste de una estructura conformada por:
  - Memoria Principal: guarda los datos y las instrucciones
  - ALU: operaciones binarias sobre los datos
  - Unidad de control: interpreta las instrucciones de la memoria y las ejecuta
  - E/S: interfaz de entradas y salidas
  - Bus de datos para comunicar CPU y Memoria



Arquitectura Von Neuman



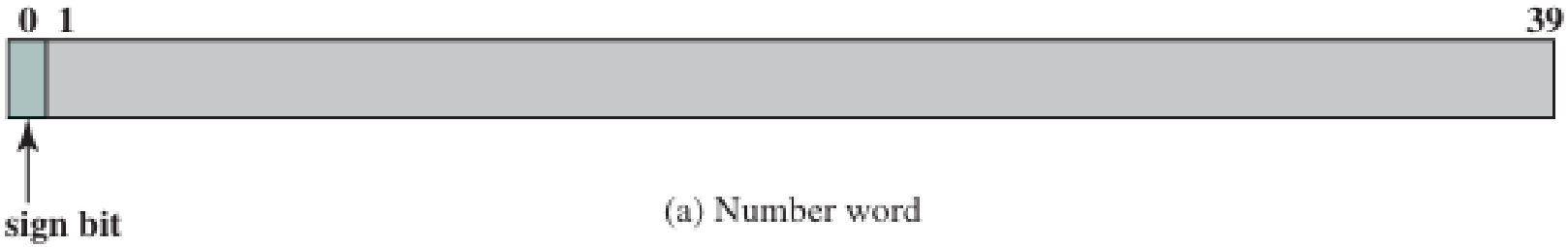
Arquitectura Von Neuman

- La memoria tiene instrucciones y datos
- Los datos e instrucciones se almacenan en binario
- La arquitectura de Von Neuman es secuencial: leer, decodificar, ejecutar
- Las arquitecturas actuales usan pipelining
- Pipelining permite hacer un *overlap* i.e. captar una nueva instrucción antes que la anterior haya concluido.

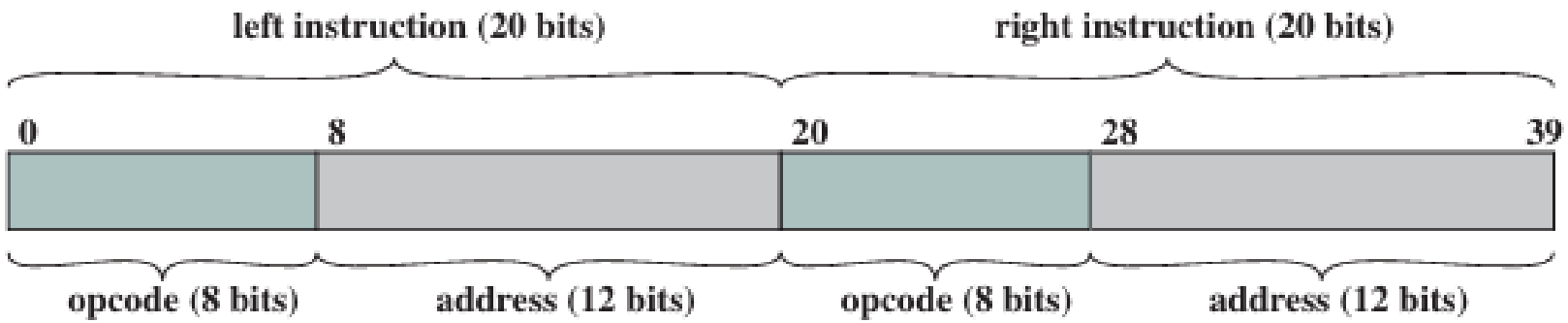
Memoria del Computador IAS I

- 4096 localidades denominadas *words*
- Cada *word* es de 40 bits
- Contiene tanto los datos como las instrucciones
- Los números usan 1 bit para el signo
- Cada *word* se subdivide en 2 instrucciones de 20 bits
- La instrucción se divide en 8 bits de /operation code/(opcode): especifica la operación
- La instrucción se divide en 12 bits que contiene la dirección en donde se aloja la instrucción o el dato

Memoria del Computador IAS



(a) Number word



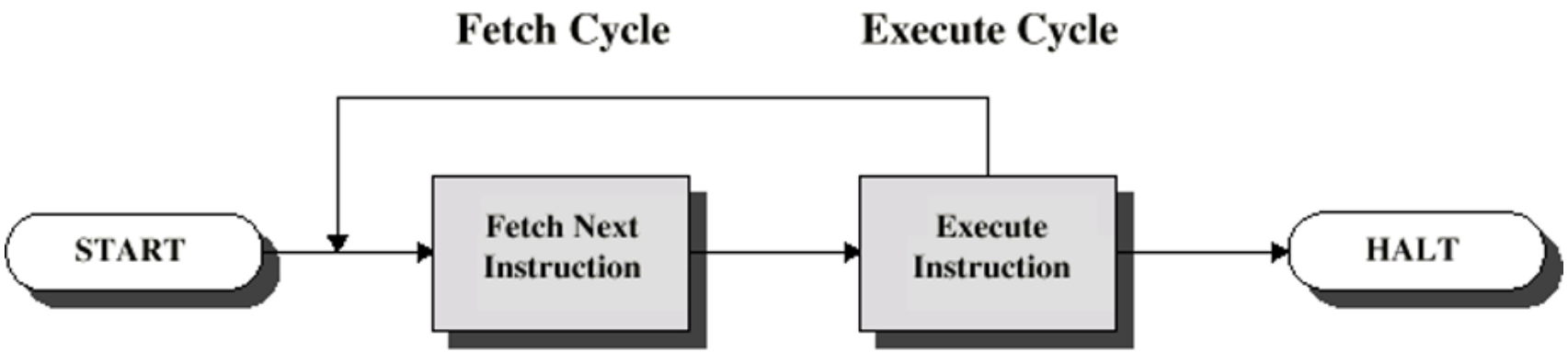
(b) Instruction word

Instruction Set Architecture (ISA) I

Define:

- Formato de Instrucciones
- Opcodes
- Registros
- Memoria e instrucciones
- El efecto de la ejecución de instrucciones sobre los registros
- El algoritmo que controla la ejecución de las instrucciones
- Se propende a que el ISA sea compatible con versiones anteriores (i.e. un programa escrito en versiones anteriores debe ser ejecutable en versiones nuevas)
- El incremento de la densidad de transistores permite ISA más complejos

Ciclo de Captación, decodificación y ejecución



Proceso de Captación I

- El contador de programa PC apunta a la siguiente instrucción: $[PC] \leftarrow [PC] + 1$
- El contenido del contador de programa se transfiere al MAR (Memory Addresds Register) $[MAR] \leftarrow [PC]$
- MAR tiene la dirección de memoria desde donde se leerá datos/instrucciones o hacia donde se escribirá.
- En un ciclo de lectura, el contenido apuntado por MAR se transfiere al MBR Memory Buffer Register $[MBR] \leftarrow [[MAR]]$
- La instrucción se pasa del MBR al IR (instruction register), donde se decodifica en el *opcode* y la dirección del dato.
- La CPU ejecuta la instrucción dictada por *opcode*.
- El *operand field* obtenido al decodificar IR puede tener una dirección o un valor constante (literal)

Proceso de Captación II

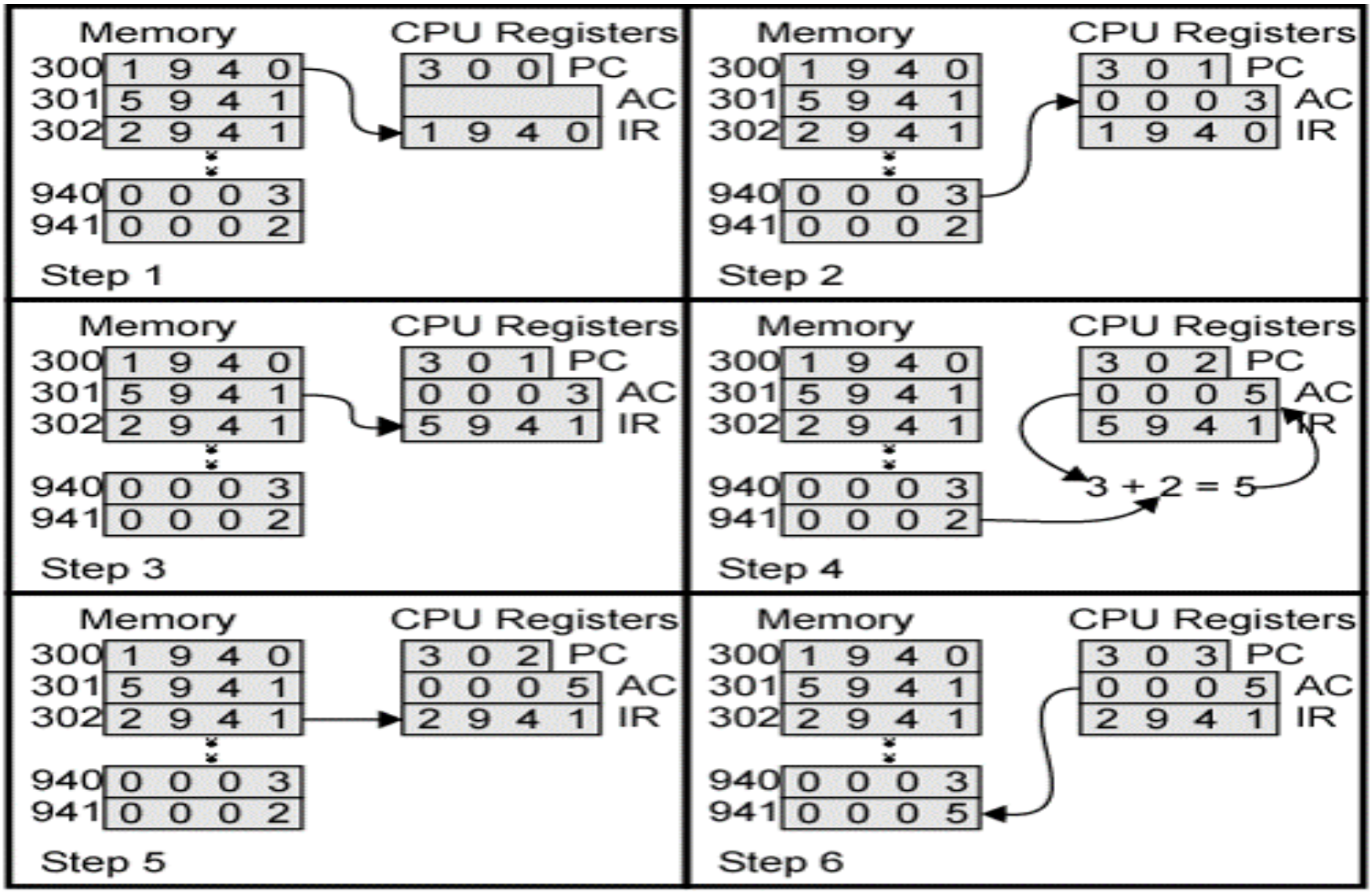
- El proceso de captación se puede escribir en notación RTL como:
  1. $[MAR] \leftarrow [PC]$
  2. $[PC] \leftarrow [PC] + 1$
  3. $[MBR] \leftarrow [[MAR]]$
  4. $[IR] \leftarrow [MBR]$
  5. $CPU \leftarrow [IR_{opcode}]$

Ejemplo I

Considere un computador de las siguientes características:

- Un único registro de acumulación AC
- La memoria es de 16 bits con 4 bits para opcode y 12 bits para direcciones de la memoria.
- ¿Cuántos Opcodes son posibles de almacenar?
- ¿Cuántas direcciones se puede alcanzar?
- El computador tiene el siguiente juego de instrucciones:
  - 0001 Cargar AC desde memoria
  - 0010 Almacenar AC en la memoria
  - 0101 Sumar al AC un dato de memoria

Ejemplo II


Memory
CPU Registers
300 1 9 4 0
301 5 9 4 1
302 2 9 4 1
940 0 0 0 3
941 0 0 0 2
3 0 0 PC
AC
1 9 4 0 IR
Step 1
3 0 1 PC
0 0 0 3 AC
1 9 4 0 IR
Step 2
3 0 1 PC
0 0 0 3 AC
5 9 4 1 IR
Step 3
3 0 2 PC
0 0 0 5 AC
5 9 4 1 IR
3 + 2 = 5
Step 4
3 0 2 PC
0 0 0 5 AC
2 9 4 1 IR
Step 5
3 0 3 PC
0 0 0 5 AC
2 9 4 1 IR
Step 6

Notación RTL Ejemplo I

Aplicando la notación `RTL` al ejemplo propuesto, este puede resolverse como:
[PC] $\leftarrow$ 300:

- [*IR*] $\leftarrow$ [300] ; El registro `IR` lee el contenido de la dirección `0x300`
- [*IR*] $\leftarrow$ 0x1940; `IR` se carga con el valor `0x1940`
- Decodificamos `IR`, observando que el opcode tiene 4 bits (i.e. 1 dígito `hex`) mientras que el operando usa 12 bits (i.e. 3 dígitos `hex`)
- [*AC*] $\leftarrow$ [940]; La instrucción `0b0001` es cargar al acumulador desde memoria
- [*AC*] $\leftarrow$ 0x0003; Se carga el acumulador con el valor 0x0003, ya que se usa el total 16 bits para las operaciones.

Notación RTL Ejemplo II

$[PC] \leftarrow 301$:

- $[IR] \leftarrow [301]$; El registro `IR` lee el contenido de la dirección `0x301`
- $[IR] \leftarrow 0x5941$
- $[AC] \leftarrow [AC] + [941]$; La instrucción `0b0101` es sumar al acumulador un dato de memoria
- $[AC] \leftarrow 0x0003 + 0x0002$; Se realiza la operación en complemento a la base
- $[AC] \leftarrow 0x0005$

Notación RTL Ejemplo III

$[PC] \leftarrow 302$:

- $[IR] \leftarrow [302]$
- $[IR] \leftarrow 0x2941$
- $[941] \leftarrow [AC]$; La instrucción 0b0010 es almacenar el Acumulador en memoria
- $[941] \leftarrow 0x0005$;

Programa Fetch en Python

```
pc = 0
mem = [0]*16
def fetch(memory):
    global pc
    mar = pc
    pc = pc +1
    mbr = memory[mar]
    ir = mbr
    cu = ir >> 8
    address = ir & 0xFF
    return (cu, address)
```

Programa Fetch en Python I

Para probar el código suponga:

- Tamaño de instrucción 12 bits
- Tamaño de la memoria de 16 localidades
- Opcode 4 bits
- Address 8 bits

Se declara 16 ubicaciones de memoria e inicializo el contador de programa en 0

```
mem = [0]*16
pc = 0
print(mem)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Asigno un valor a la primera instrucción. Donde los 4 primeros dígitos son opcode

Programa Fetch en Python II

```
mem[0] = 0b011000001010
mem[1] = 0b100011111111
print(mem)
```

```
[1546, 2303, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
opCode, address = fetch(mem)
print(f"pc = {pc-1}\nopcode = {opCode}\nOperand = {address}")
opCode, address = fetch(mem)
print(f"pc = {pc-1}\nopcode = {opCode}\nOperand = {address}")
```

```
pc = 0
opcode = 6
Operand = 10
pc = 1
opcode = 8
Operand = 255
```

Simulación ciclos Von Neuman

Adaptar el código fetch para resolver las operaciones planteadas en el ejemplo. Para esto se necesitaría escribir una rutina de *decode* para decodificar las instrucciones y otra de ejecución que permitan obtener los resultados.

- Memoria de 16 bits
- 4 bits opcode
- 12 bits address
- Un único registro de acumulación AC

opcode	Función
0001	Cargar AC desde Memoria
0010	Almacenar AC en Memoria
0101	Sumar AC un dato de Memoria