

# Ejecución de programas simples en una arquitectura simple

Organización de computadoras

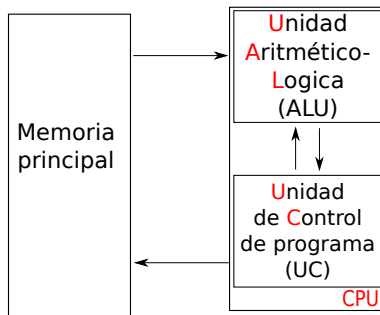
Universidad Nacional de Quilmes

26 de agosto de 2013

# Repaso

- ➊ Lógica proposicional ¿Para qué?
- ➋ Puertas
  - ➊ ¿Qué son?
  - ➋ Las elementales: AND, OR, NOT
  - ➌ Algunas derivadas: NAND, NOR, XOR
- ➌ Circuitos
  - ➊ ¿Qué son?
  - ➋ ¿Cómo se construyen?
    - ➊ Enunciado
    - ➋ Tabla de verdad
    - ➌ SOP
    - ➍ Circuito
- ➍ Circuitos comunes
- ➎ Circuitos Aritméticos

# Arquitectura de Von Neumann



# Unidad aritmético lógica

## ALU (Unidad aritmético lógica)

Dispositivo que realiza las operaciones aritméticas y lógicas sobre los datos de entrada que se le proveen.

¿Cómo se usa?

- 1 La UC (unidad de control) suministra los operandos a la ALU
- 2 La UC indica a la ALU la operación a llevar a cabo.
- 3 La ALU realiza la operación.
- 4 La UC toma el resultado.

# Registros internos

## Registro

Espacio de almacenamiento interno a la CPU. Es la tecnología de almacenamiento mas rápida del sistema de cómputos.

- Para ser procesado, todo dato debe alojarse en un **registro** interno a la CPU
- **Algunos** están disponibles para ser usados por los programas.
- Otros están **reservados** para uso interno de la Unidad de Control

# Código binario

## Código binario







Las computadoras tienen la capacidad de traducir una cadena binaria en una acción determinada



Los programas deben estar codificados en binario

## Código binario

### Ejemplo: decodificando una receta

|     |   |                |
|-----|---|----------------|
| 001 |  | agregar huevo  |
| 000 |  | mezclar        |
| 010 |  | agregar harina |
| 000 |  | mezclar        |
| 110 |  | agregar leche  |
| 000 |  | mezclar        |



# Código binario

## Código Fuente

Código comprensible por un humano

## Código Máquina

Código directamente interpretable por la CPU

## Código binario

¿Cuándo se **ensambla** (código fuente → código máquina)?

¿Cuándo se **desensambla** (código máquina → código fuente)?

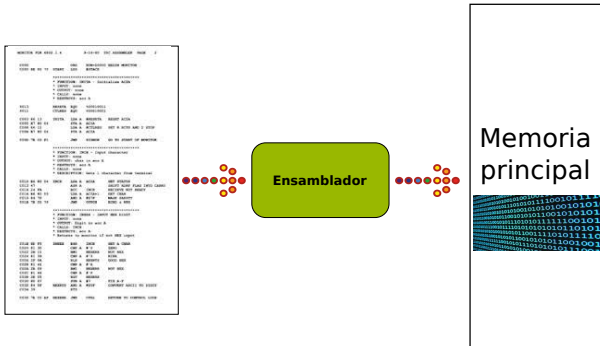
# Ciclo de vida de un programa

## Ciclo de vida de un programa

El **programador** escribe el programa



El ensamblador lo traduce a **código máquina** y lo carga en memoria



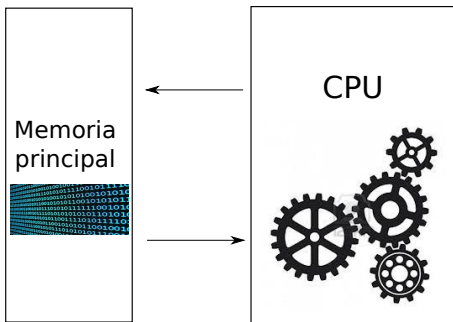
## Ciclo de vida de un programa

El **usuario** pide la ejecución del programa



## Ciclo de vida de un programa

La CPU ejecuta el programa



# Jugando a la panadería



# Simulación: La Panadería

## Reglas básicas



- Las recetas **se codifican para recordarlas**
- Las recetas se decodifican para cocinarlas
- En ambos casos se sigue un manual (especificación)
- El jefe de cocina tiene dos ayudantes:
  - El responsable de los ingredientes
  - El responsable de los instrumentos

# Simulación: La Panadería

## Reglas básicas



- Las recetas se codifican para recordarlas
- Las recetas **se decodifican para cocinarlas**
- En ambos casos se sigue un manual (especificación)
- El jefe de cocina tiene dos ayudantes:
  - El responsable de los ingredientes
  - El responsable de los instrumentos

# Simulación: La Panadería












## Reglas básicas



- Las recetas se codifican para recordarlas
- Las recetas se decodifican para cocinarlas
- En ambos casos **se sigue un manual** (especificación)
- El jefe de cocina tiene dos ayudantes:
  - El responsable de los ingredientes
  - El responsable de los instrumentos

# Simulación: La Panadería

## El manual

|   |  |             |   |                               |             |
|---|--|-------------|---|-------------------------------|-------------|
|  | <b>Agregar<br/>1 huevo</b>             | <b>0000</b> |  | <b>Batir</b>                  | <b>0101</b> |
|  | <b>Agregar 1<br/>taza harina</b>       | <b>0001</b> |  | <b>Mezclar</b>                | <b>0110</b> |
|  | <b>Agregar 1<br/>taza azucar</b>       | <b>0010</b> |  | <b>Hornear<br/>suave 10'</b>  | <b>0111</b> |
|  | <b>Agregar 1 cda<br/>polvo hornear</b> | <b>0011</b> |  | <b>Hornear<br/>medio 10'</b>  | <b>1000</b> |
|  | <b>Agregar 1<br/>cda aceite</b>        | <b>0100</b> |  | <b>Hornear<br/>fuerte 10'</b> | <b>1001</b> |
|  | <b>Beep</b>                            | <b>1010</b> |   |                               |             |

# Simulación: La Panadería

## Codificación



|                            |        |
|----------------------------|--------|
| ➊ Agregar 1 huevo          | ➊ 0000 |
| ➋ Agregar 1 taza de harina | ➋ 0001 |
| ➌ mezclar                  | ➌ 0110 |

# Simulación: La Panadería

## Codificación



|                            |        |
|----------------------------|--------|
| ➊ Agregar 1 huevo          | ➊ 0000 |
| ➋ Agregar 1 taza de harina | ➋ 0001 |
| ➌ mezclar                  | ➌ 0110 |

# Simulación: La Panadería


## Decodificación





1 0000

2 0001

3 0110

1 Agregar 1 huevo  El cocinero en jefe delega al responsable de los ingredientes

2 Agregar 1 taza de harina  El cocinero en jefe delega al responsable de los ingredientes

3 mezclar  El cocinero en jefe delega al responsable de los instrumentos

# Simulación: La Panadería


## Decodificación




1 0000

2 0001

3 0110

1 Agregar 1 huevo  El cocinero en jefe delega al responsable de los ingredientes

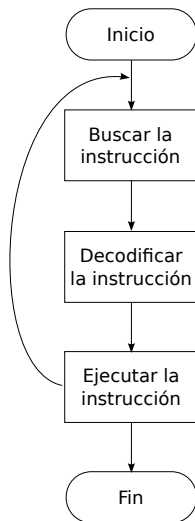
2 Agregar 1 taza de harina  El cocinero en jefe delega al responsable de los ingredientes

3 mezclar  El cocinero en jefe delega al responsable de los instrumentos



# Ciclo de ejecución de una instrucción

## Ciclo de ejecución de una instrucción



# Arquitectura Q1

# Arquitectura Q1

- Tiene 8 registros de uso general de 16 bits: R0..R7
- Tiene instrucciones de 2 operandos:

| instrucción | sintaxis            | efecto   |
|-------------|---------------------|--|
| ADD         | ADD destino, origen | $\text{destino} \leftarrow \text{destino} + \text{origen}$       |
| SUB         | SUB destino, origen | $\text{destino} \leftarrow \text{destino} - \text{origen}$       |
| MUL         | MUL destino, origen | $(R7, \text{destino}) \leftarrow \text{destino} * \text{origen}$ |
| DIV         | DIV destino, origen | $\text{destino} \leftarrow \text{destino} \% \text{origen}$      |
| MOV         | MOV destino, origen | $\text{destino} \leftarrow \text{origen}$                        |

- Los operandos pueden ser **variables** (alguno de los registros) o **constantes**.

# Arquitectura **Q1**: modos de direccionamiento

## **Modo de direccionamiento**

Mecanismo por el cual la unidad de control obtiene el operando requerido

**Q1** permite 2 modos de direccionamiento:

- modo **registro**: el valor buscado está en un registro
- modo **inmediato**: el valor buscado está codificado dentro de la instrucción (constante)

# Arquitectura **Q1**: modos de direccionamiento

Modo de direccionamiento Inmediato

MOV R0, 0x3456

ADD R6, 0xFEFE

# Arquitectura **Q1**: modos de direccionamiento

Modo de direccionamiento Registro

MOV **R0**, 0x3456

ADD **R6**, **R1**

# Arquitectura Q1: formato de instrucciones

## Formato de instrucción

Define la organización de los bits dentro de una instrucción, en términos de las partes que la componen. Debe incluir el **código de la operación** y los **operandos**

Cuando se tiene una cadena así:

```
0000100000000000000001111000011110000111100001111
```



Se separan las partes así

```
0000100000000000000001111000011110000111100001111
```



# Arquitectura **Q1**: formato de instrucciones

0000100000000000000001111000011110000111100001111



0000100000000000000001111000011110000111100001111

|                |                      |                     |                           |                          |
|----------------|----------------------|---------------------|---------------------------|--------------------------|
| Cod_Op<br>(4b) | Modo Destino<br>(6b) | Modo Origen<br>(6b) | Operando Destino<br>(16b) | Operando Origen<br>(16b) |
|----------------|----------------------|---------------------|---------------------------|--------------------------|

# Arquitectura **Q1**: Códigos de Operación

| Operación | CodOp |
|-----------|-------|
| MUL       | 0000  |
| MOV       | 0001  |
| ADD       | 0010  |
| SUB       | 0011  |
| DIV       | 0111  |

# Arquitectura **Q1**: Códigos de los modos de direccionamiento

| Modo      | Codificación |
|-----------|--------------|
| Inmediato | 000000       |
| Registro  | 100rrr       |

donde *rrr* es una codificación (en 3 bits) del número de registro.

# Arquitectura Q1: formato de instrucciones

|                |                      |                     |                           |                          |
|----------------|----------------------|---------------------|---------------------------|--------------------------|
| Cod_Op<br>(4b) | Modo Destino<br>(6b) | Modo Origen<br>(6b) | Operando Destino<br>(16b) | Operando Origen<br>(16b) |
|----------------|----------------------|---------------------|---------------------------|--------------------------|

Los campos de los operandos Destino y Origen...

- contienen valores constantes (si el modo respectivo es *inmediato*)
- o no existen (si el modo respectivo es *registro*).

## Arquitectura Q1: Ejemplos

### Ejercicio: Ensamblar MOV R1,0x0003

|                     |                                       |
|---------------------|---------------------------------------|
| Efecto              | $R1 \leftarrow 0x0003$                |
| Código de operación | 0001                                  |
| Modo Destino        | R1 está en modo registro: 100rrr      |
| Modo Origen         | 0x0003 está en modo inmediato: 000000 |



00011000010000000000000000000011

# Arquitectura Q1: Ejemplos

## Ejercicio: ensamblar MOV R1,R6

|                     |                                  |
|---------------------|----------------------------------|
| Efecto              | $R1 \leftarrow R6$               |
| Código de operación | 0001                             |
| Modo Destino        | R1 está en modo registro: 100001 |
| Modo Origen         | R6 está en modo registro: 100110 |



0001100001100110

# Arquitectura Q1: Ejercicios

## Ejercicios

- 1 Hacer un programa que multiplique por 12 el valor de R0.
- 2 Hacer un programa que sume R0 con R1 y guarde el resultado en R2

# Arquitectura **Q1**: Ejercicios

## Ejercicio: ensamblar el siguiente programa

```
SUB R0, R1  
ADD R2, R0  
DIV R2, 7
```

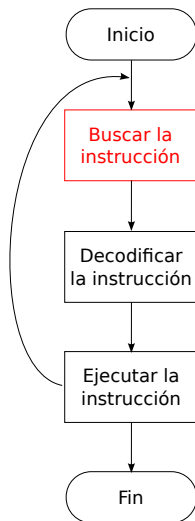


# Arquitectura Q1: Ejercicios

## Ejercicio: Desensamblar

| cadena (hexa) | Codop | Modo Destino | Modo Origen | Origen | Destino |
|---------------|-------|--------------|-------------|--------|---------|
| 1821          | MOV   | Registro     | Registro    | R1     | R0      |
| 0860          | MUL   | Registro     | Registro    | R1     | R0      |
| 28400005      | ADD   | Registro     | Inmediato   | 0005   | R1      |

# Ciclo de ejecución de una instrucción



## Ciclo de ejecución de una instrucción

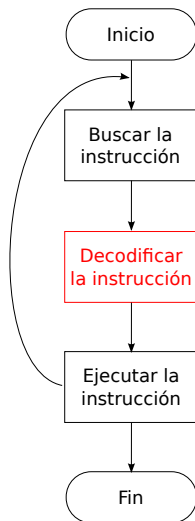
### Búsqueda de la instrucción

- 1 La UC pide a la memoria un conjunto de bits
- 2 La memoria le envía el sector pedido

La UC **sabe** que el sector pedido contiene una instrucción

La memoria **no lo sabe**

# Ciclo de ejecución de una instrucción

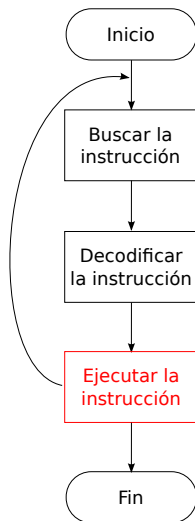


# Ciclo de ejecución de una instrucción

## Decodificación de la instrucción

- 1 La UC extrae la operación de determinados bits de la cadena leída

## Ciclo de ejecución de una instrucción



# Ciclo de ejecución de una instrucción

## Ejecución de la instrucción

- 1 La UC traduce la operación en señales eléctricas a:
  - la ALU (dándole parámetros y obteniendo resultados)
  - la memoria (pidiendole la lectura o escritura de bits)
  - los registros (poniendoles valor o leyendo el contenido)

# Conclusiones

## Conclusiones



- Para ejecutar un programa se lo debe **Ensamblar**
- Hemos hecho tareas manuales (humanas):
  - 1 **Desensamblar**
  - 2 **Representar números en binario**



# Conclusiones

¿Preguntas?



Buzón de sugerencias