

# Microarquitectura

## Organización del Computador I

Paula Verghelet

Departamento de Computación  
Facultad de Ciencias Exactas y Naturales  
Universidad de Buenos Aires

1er cuatrimestre - 2018

# Agenda

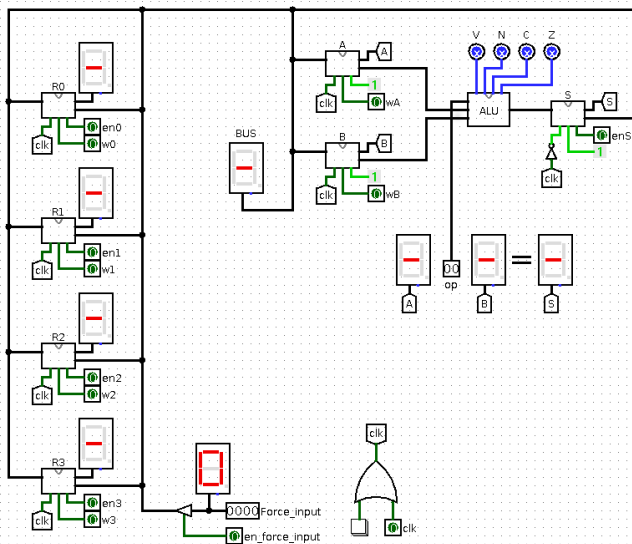
- Introducción
- Lenguaje y Notación
- Ejercicios

Instrucción → ??? → Circuito

### Microinstrucciones

- Cada instrucción está compuesta por microinstrucciones. Al igual que las partes del ciclo de instrucción.
  - Estas microinstrucciones realizan pasos muy simples.
  - Poner cosas en registros, activar circuitos, prender/apagar líneas.
  - Entonces, la UC (**U**nidad de **C**ontrol) tiene que ejecutar estas microinstrucciones.
  - Ejemplo microprograma: FETCH, visto en la teórica.
- 
- ¿Y cómo hace?
  - Mediante señales de control.

# ¿Recuerdan el Taller Lógica Digital?

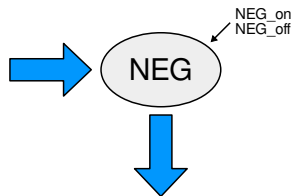
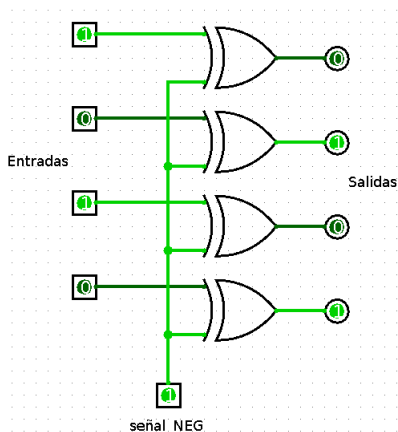


## El lenguaje y Notación : Componentes

- Definimos **componentes** como circuitos con entradas, salidas y señales
- Las **señales** son entradas que modifican el comportamiento de los circuitos
- Las señales **se activan** según como indique el *microprograma*
- El objetivo aquí es escribir microprogramas que **describan comportamientos**
- Estos serán listas de asignaciones entre registros y activación de señales, se realiza **un evento por ciclo de clock**

# El lenguaje y Notación : Componentes

## Ejemplo de un componente negador



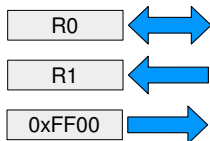
NEG\_on : niega los bits de la entrada y los pone a la salida

NEG\_off : pasa los bits sin modificarlos

## El lenguaje y Notación : Registros

- Existen registros que almacenan conjuntos de valores

Ejemplos



- Los registros pueden ser usados por completo o parte de ellos

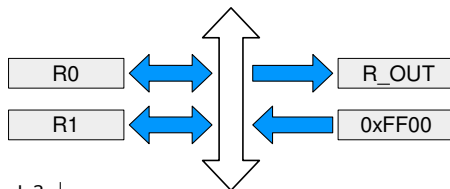
Ejemplos

- R8 : Registro R8
- R0[0] : Bit 0 del registro R0
- R2[3:2] : Del bit 2 al bit 3 del registro R2

# El lenguaje y Notación : Líneas

- Los datos se mueven por caminos (líneas)
- Podemos mover un dato de un registro a otro si hay un camino directo entre ellos
- Podemos asignar un valor constante almacenado a un registro

## Ejemplos

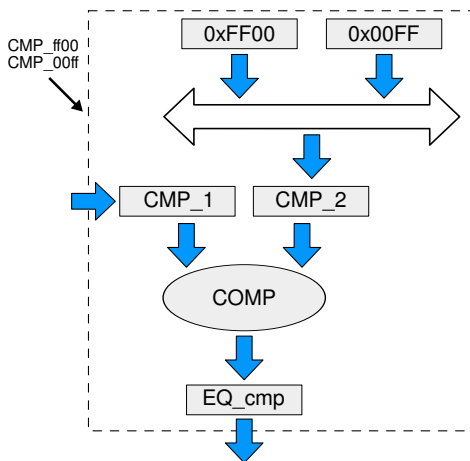


	¿vale?	
R1:= R0	✓	Copia el contenido de R0 en R1
R_OUT:=R0	✓	Copia el contenido de R0 en R_OUT
R0:=0xFF00	✓	Copia la constante 0xFF00 a R0
R0:=R_OUT	×	El registro R_OUT es de solo escritura
R1:=x	×	No es posible asignar una constante



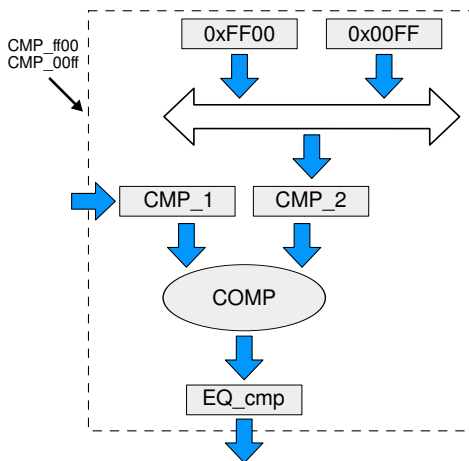
## El lenguaje y Notación : nuevos componentes

Podemos, además, construir componentes que a su vez tengan sus propias señales



## El lenguaje y Notación : nuevos componentes

Podemos, además, construir componentes que a su vez tengan sus propias señales



Ejemplo de uso:

```
CMP_1 := R3
CMP_FF00
if EQ_cmp = 0
    R_OUT := R3
else
    R_OUT := R2
endif
```

Podemos analizar el valor de un bit y actuar en consecuencia

# Ejercicio 1

Se cuenta con los siguientes circuitos:

- Una ALU con 2 registros de 16 bits ( $ALU\_IN1$  y  $ALU\_IN2$ ) que usa de entradas y 5 registros que usa de salida:  $ALU\_OUT$  de 16 bits y ( $ALU\_Z$ ,  $ALU\_N$ ,  $ALU\_C$  y  $ALU\_V$ ) de 1 bit.

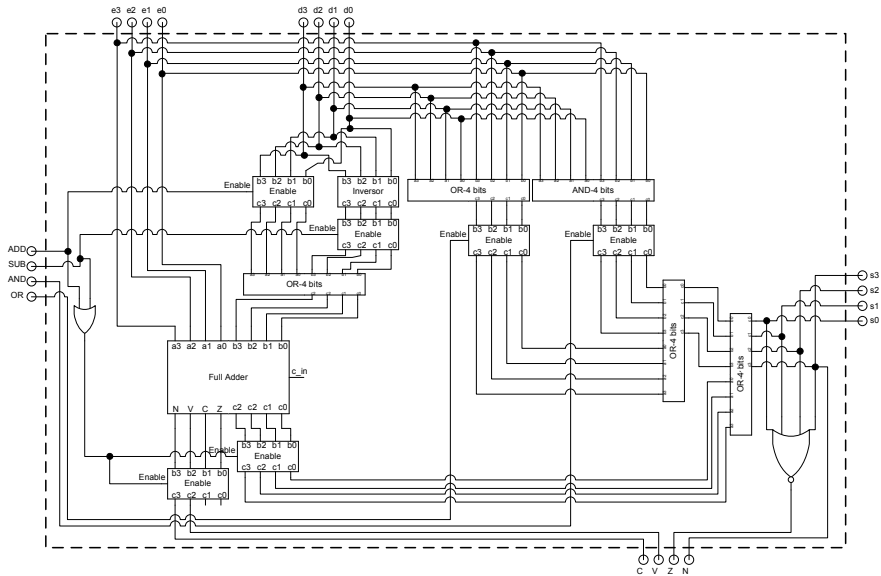
Sus señales de control son:

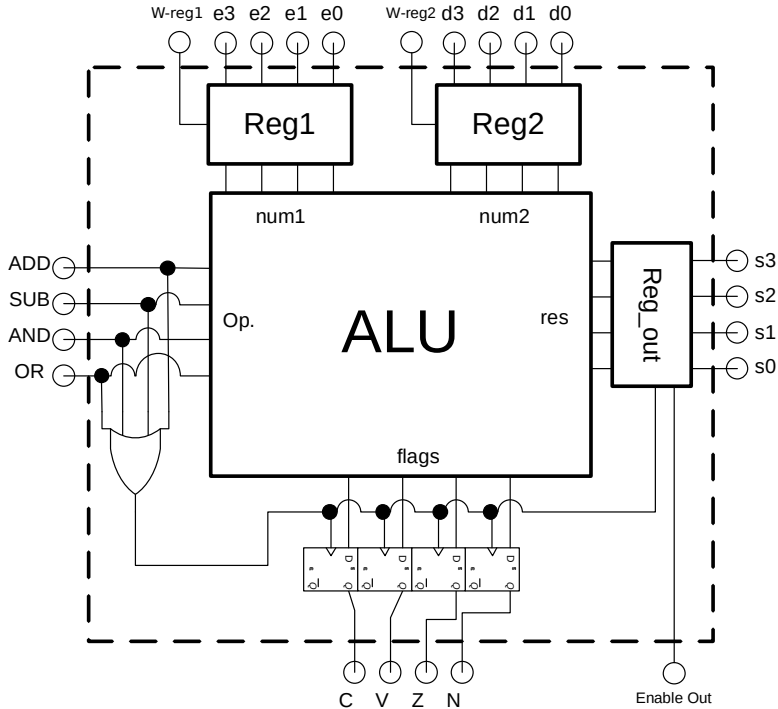
Señal	Efecto
$ALU_{add}$	$ALU\_OUT := ALU\_IN1 + ALU\_IN2$
$ALU_{sub}$	$ALU\_OUT := ALU\_IN1 - ALU\_IN2$
$ALU_{neg}$	$ALU\_OUT := - ALU\_IN1$
$ALU_{and}$	$ALU\_OUT := ALU\_IN1 \text{ AND } ALU\_IN2$
$ALU_{not}$	$ALU\_OUT := \text{NOT } ALU\_IN1$

- Un extensor de signo complemento a 2 ( $SIGN\_EXT$ ) con un registro de entrada de 8 *bits* ( $EXT\_IN$ ) y un registro de salida de 16 *bits* ( $EXT\_OUT$ ).

Sus señales de control son:

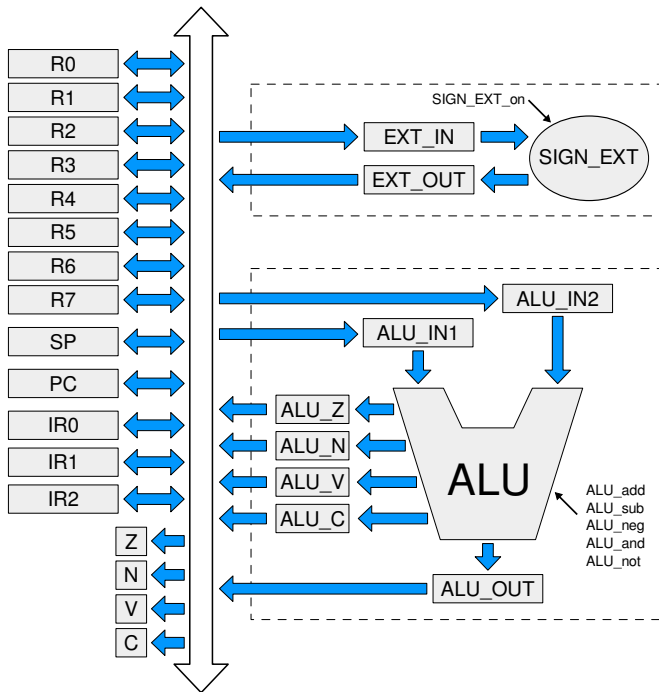
Señal	Efecto
$SIGN\_EXT_{on}$	activa la operación de extensión de signo de 8 <i>bits</i> a 16 <i>bits</i>





# Ejercicio 1

- 1 Suponiendo que se encuentra resuelta la decodificación y el acceso a memoria de la máquina, diseñar el camino de datos de la arquitectura de la máquina ORGA1. No dibujar la unidad de control para simplificar el diagrama.



# Ejercicio 1

## Características

- Registros de 16 bits: R0, R1, R2, R3, R4, R5, R6, R7, SP, PC, IR0, IR1, IR2, EXT\_OUT, ALU\_IN1, ALU\_IN2, ALU\_OUT
- Registro de 8 bits: EXT\_IN
- Registros de 1 bit: Z, N, V, C, ALU\_Z, ALU\_N, ALU\_V, ALU\_C
- Bus interno: 16 líneas
- Los flags están conectados a las 4 líneas menos significativas del bus
- El registro EXT\_IN está conectado a las 8 líneas menos significativas del bus



# Ejercicio 1

- 2 Indicar cuál es la secuencia de señales (o microoperaciones) que debe realizar la unidad de control para ejecutar las siguientes instrucciones:
- MOV R5, R1
  - AND R7, R1
  - JE 0xFF

# Ejercicio 1

## ■ MOV R5,R1

1. R5 := R1

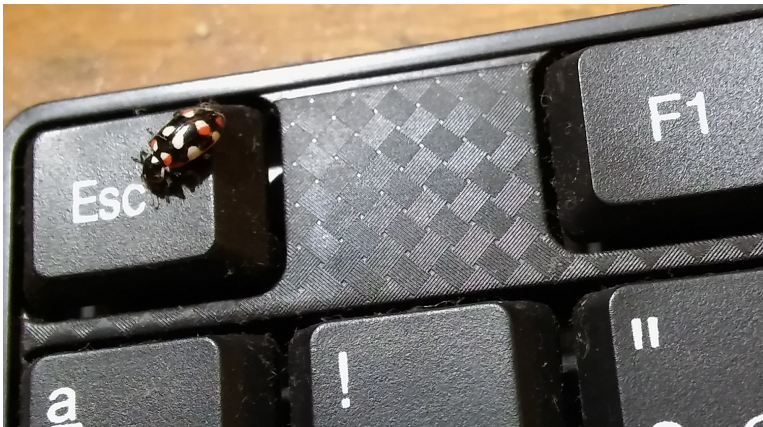
## ■ AND R7, R1

1. ALU\_IN1 := R7
2. ALU\_IN2 := R1
3. ALU<sub>and</sub>
4. R7 := ALU\_OUT
5. Z := ALU\_Z
6. N := ALU\_N
7. C := ALU\_C
8. V := ALU\_V

## ■ JE 0xFF

1. if Z = 1
2. EXT\_IN := IR0 [7:0]
3. SIGN\_EXT\_on
4. ALU\_IN\_1 := PC
5. ALU\_IN\_2 := EXT\_OUT
6. ALU<sub>add</sub>
7. PC := ALU\_OUT
8. endif

# ¿Intervalo?



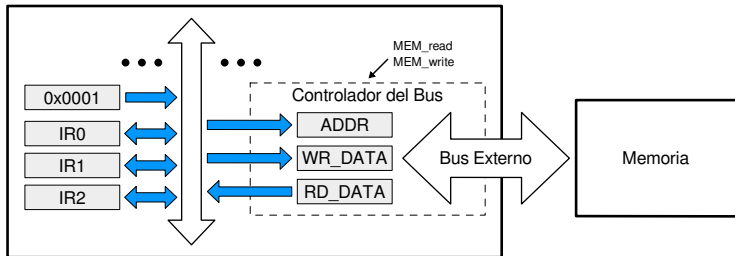
## Ejercicio 2

Se cuenta con una memoria con palabra y direccionamiento de 16 *bits*. Posee 2 registros de entrada de 16 *bits* (ADDR, WR\_DATA) y 1 de salida de 16 *bits* (RD\_DATA).

Sus señales de control son:

- MEM\_WRITE: Activa la microoperación de escritura del contenido del registro WRT\_DATA en la dirección de memoria indicada por el ADDR
  - MEM\_READ: Activa la microoperación de lectura del contenido de la dirección de memoria indicada por el ADDR, colocando el valor en el registro RD\_DATA.
- 1 Extender el camino de datos de la arquitectura de la máquina ORGA1. No dibujar la unidad de control para simplificar el diagrama.
  - 2 ¿Qué componentes del camino de datos se encuentran dentro del CPU y fuera de él?
  - 3 Indicar cuál es la secuencia de señales (o microoperaciones) que debe realizar la unidad de control para ejecutar las siguientes instrucciones:
    - MOV R2, R5
    - MOV R2, [R5]
    - MOV R2, [0xFF00]
    - MOV [0xFF00], [0xFF01]
  - 4 Describa la secuencia de microoperaciones que realiza la unidad de control para realizar un *fetch* de una instrucción de la máquina ORGA1.

## Ejercicio 2 - Camino de datos



## Ejercicio 2 - Microoperaciones

### ■ MOV R2, R5

1.  $R2 := R5$

### ■ MOV R2, [R5]

1. ADDR := R5
2. MEM\_READ
3.  $R2 := RD\_DATA$

### ■ MOV R2, [0xFF00]

1. ADDR := IR1
2. MEM\_READ
3.  $R2 := RD\_DATA$

### ■ MOV [0xFF00], [0xFF01]

1. ADDR := IR2
2. MEM\_READ
3.  $WRT\_DATA := RD\_DATA$
4. ADDR := IR1
5. MEM\_WRITE

## Ejercicio 2 - *fetch*

1. ADDR := PC
2. MEM.READ
3. IR0 := RD\_DATA // cargo instrucción en IR0
4. ALU\_IN1 := PC
5. ALU\_IN2 := 0x0001 // cargo **CTE almacenada**
6. ALU.add // incremento PC
7. PC := ALU.OUT // guardo PC incrementado
8. IF IR0[11] = 0 //Si el operando destino es Inmediato, Directo o Indirecto.
9.     // Repito pasos 1 a 7 (cargo operando en IR1)
10.    IF IR0[5] = 0 //Si el operando fuente es Inmediato, Directo o Indirecto.
11.       // Repito pasos 1 a 7 (cargo operando en IR2)
12.       ELSE IF IR0[3] = 1 //Si el operando fuente es Indexado
13.          // Repito pasos 1 a 7 (cargo operando en IR2)
14.       ELSE IF IR0[9] = 1 //Si el operando destino es Indexado
15.          // Repito pasos 1 a 7 (cargo operando en IR1)
16.       IF IR0[5] = 0 //Si el operando fuente es Inmediato, Directo o Indirecto.
17.          // Repito pasos 1 a 7 (cargo operando en IR2)
18.       ELSE IF IR0[3] = 1 //Si el operando fuente es Indexado
19.          // Repito pasos 1 a 7 (cargo operando en IR2)
20.       ELSE IF IR0[5] = 0 //Si el operando fuente es Inmediato, Directo o Indirecto.
21.          // Repito pasos 1 a 7 (cargo operando en IR1)
22.       ELSE IF IR0[3] = 1 //Si el operando fuente es Indexado
23.          // Repito pasos 1 a 7 (cargo operando en IR1)

## Ejercicio 3

La computadora STACK1 es una máquina de pila con direccionamiento a byte, tamaño de palabra de 16 bits y direcciones de memoria de 12 bits. Trabaja con aritmética complemento a 2 de 16 bits. Posee el siguiente set de instrucciones:

Instrucción	CodOp	Significado
PUSH [M]	0000	push [X]
POP [M]	0001	[X] := pop
ADD	0010	push(pop+pop)
SUB	0011	push(pop-pop)
JUMP	0100	PC := pop (sólo los 12 bits menos significativos)
SKIP_N	0101	ignora la próxima instrucción si top es < 0
SKIP_Z	0110	ignora la próxima instrucción si top es 0
SKIP_GE	0111	ignora la próxima instrucción si top es $\geq 0$

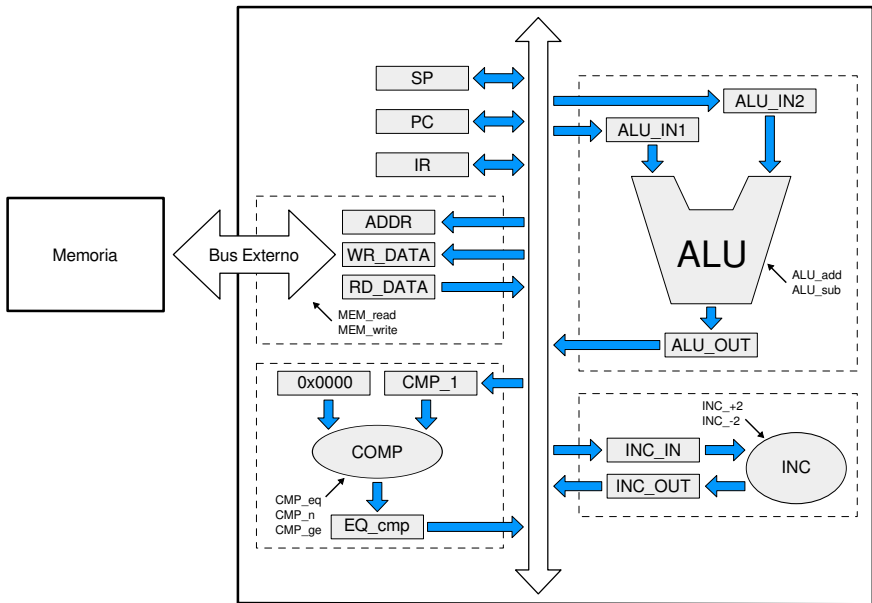
El formato de instrucción de STACK1 es el que sigue:

CodOp	Dirección
4 bits	12 bits



## Ejercicio 3

- 1 Definir el camino de datos y la organización del CPU de STACK1 para soportar la implementación de, al menos, estas instrucciones.
- 2 Describa la secuencia de microoperaciones que realiza la unidad de control para realizar un *fetch* de una instrucción.
- 3 Implementar las siguientes instrucciones:
  - I) JUMP
  - II) SKIP\_Z
  - III) PUSH [X]
  - IV) ADD



## Ejercicio 3 - Camino de datos

- Se utiliza un circuito incrementador con 2 señales:  
INC<sub>+2</sub>: suma 2 a la entrada  
INC<sub>-2</sub>: resta 2 a la entrada
- Registros ADDR, INC\_IN, INC\_OUT, SP y PC de 12 bits
- Registros IR, CMP\_1, ALU\_IN1, ALU\_IN2, ALU\_OUT, WR\_DATA y RD\_DATA de 16 bits
- Los buses INTERNO y EXTERNO de 16 bits
- EQ\_CMP es de 1 bit
- Las 12 líneas de los registros correspondientes están conectadas a las líneas menos significativas del BUS
- El SP apunta a la primer dirección libre

## Ejercicio 3 - *fetch*

1. ADDR := PC
2. MEM\_READ
3. IR := RD\_DATA // cargo el IR
4. INC\_IN := PC
5. INC<sub>→</sub>+2
6. PC := INC\_OUT // incremento PC

## Ejercicio 3 - microoperaciones

### ■ JUMP

1. INC\_IN := SP
2. INC<sub>+2</sub>
3. SP := INC\_OUT
4. ADDR := SP
5. MEM\_READ
6. PC := RD\_DATA[11:0]

### ■ SKIP\_Z

1. INC\_IN := SP
2. INC<sub>+2</sub>
3. ADDR := INC\_OUT
4. MEM\_READ
5. CMP\_1 := RD\_DATA
6. CMP\_eq
7. if EQ\_cmp = 0
8.   INC\_IN := PC
9.   INC<sub>+2</sub>
10.   PC := INC\_OUT
11. endif

## Ejercicio 3 - microoperaciones

### ■ PUSH [X]

1. ADDR := IR[11:0] // leo la direccion en el operando, [X]
2. MEM\_READ
3. WRT\_DATA := RD\_DATA //leo el valor que voy a escribir en el tope de la pila , memoria
4. ADDR := SP // voy a escribir en el tope de la pila
5. MEM\_WRITE //escribo el contenido de X en el tope de la pila
6. INC\_IN := SP
7. INC\_<sub>--2</sub>
8. SP := INC\_OUT

### ■ ADD

1. INC\_IN := SP
2. INC\_<sub>+2</sub>
3. SP := INC\_OUT
4. ADDR := SP
5. MEM\_READ
6. ALU\_IN1 := RD\_DATA // primer operando
7. INC\_IN := SP
8. INC\_<sub>+2</sub>
9. SP := INC\_OUT
10. ADDR := SP
11. MEM\_READ
12. ALU\_IN2 := RD\_DATA // segundo operando
13. ALU\_<sub>-add</sub>
14. WRT\_DATA := ALU\_OUT
15. ADDR := SP
16. MEM\_WRITE // push resultado
17. INC\_IN := SP
18. INC\_<sub>--2</sub>
19. SP := INC\_OUT

# ¿Preguntas?



# ¿Preguntas?

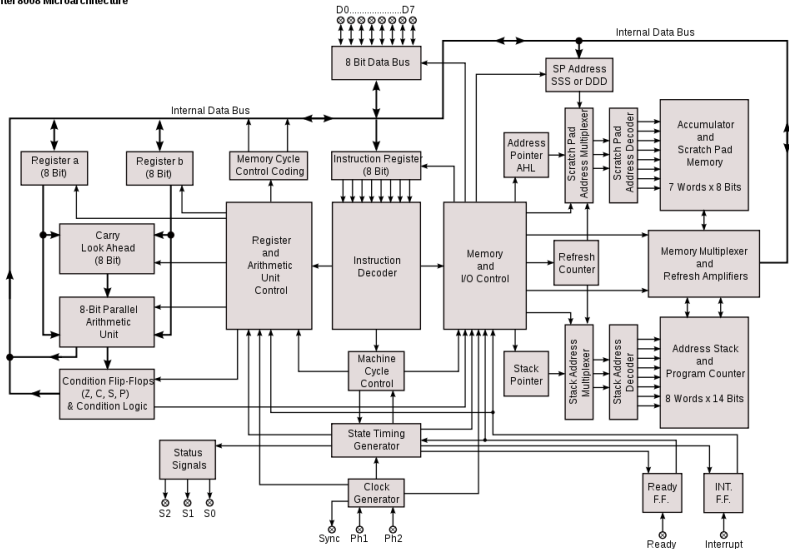


La Hormiga Atómica

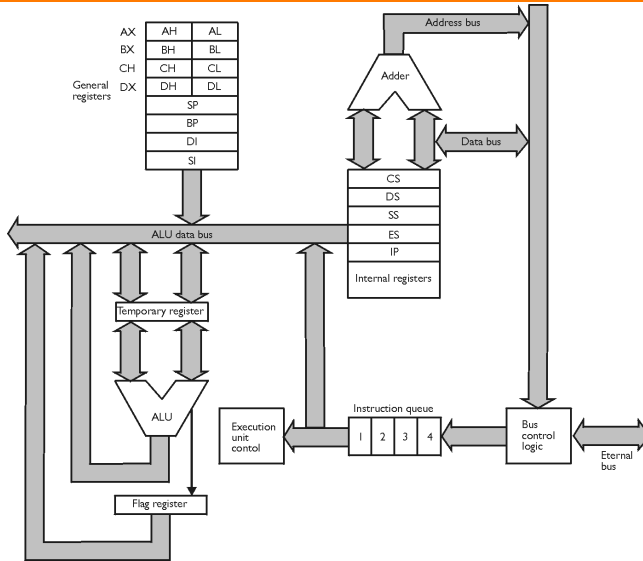


# Microarchitecture 8008 (1972)

Intel 8008 Microarchitecture

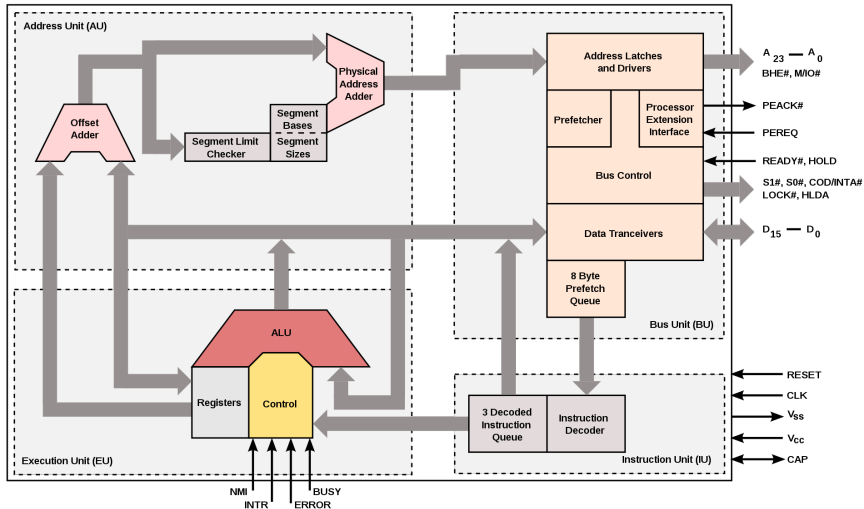


# Microarchitecture 8086 (1979)



# Microarchitecture 80286 (1982)

Intel 80286 architecture



# Microarchitecture 80386 (1986)

