Clase 08 - Subrutinas

IIC2343 - Arquitectura de Computadores

Profesor:
- Felipe Valenzuela González
Correo:
frvalenzuela@alumni.uc.cl

Resumen de la clase pasada

Salto Incondicional: Instrucciones (ejemplo)

- La limitante es el hecho de **no iterar**
- Las instrucciones que nos permiten cambiar la dirección de código las llamaremos **saltos**
- En particular en el ejemplo tenemos un **salto incondicional**

Dirección	Instrucción	Operandos
0x00	MOV	A,0
0x01	MOV	B,1
0x02	ADD	A,B
0x03	ADD	B,A
0x04	JMP	0x02

Imagen Referencia: https://github.com/IIC2343/Syllabus-anteriores/blob/main/Apuntes%20(viejos)/5%20-%20Saltos%20y%20Subrutinas.pdf

Salto Incondicional: Label

- Es un **indicador** que se puede agregar en una línea del código assembly para referirse a la **dirección de memoria** asociada a esa línea

Dirección	Label	Instrucción	Operandos
0x00		MOV	A,0
0x01		MOV	B,1
0x02	start:	ADD	A,B
0x03		ADD	B,A
0x04		JMP	start

Imagen Referencia: https://github.com/IIC2343/Syllabus-anteriores/blob/main/Apuntes%20(viejos)/5%20-%20Saltos%20y%20Subrutinas.pdf

Salto Incondicional: Implementación

- La respuesta es modificar el resgistro del **Program Counter**
- Esta conexión va desde el literal que viene de la **Instruction Memory**
- Agregamos una nueva señal de control **Lpc** (Load PC)

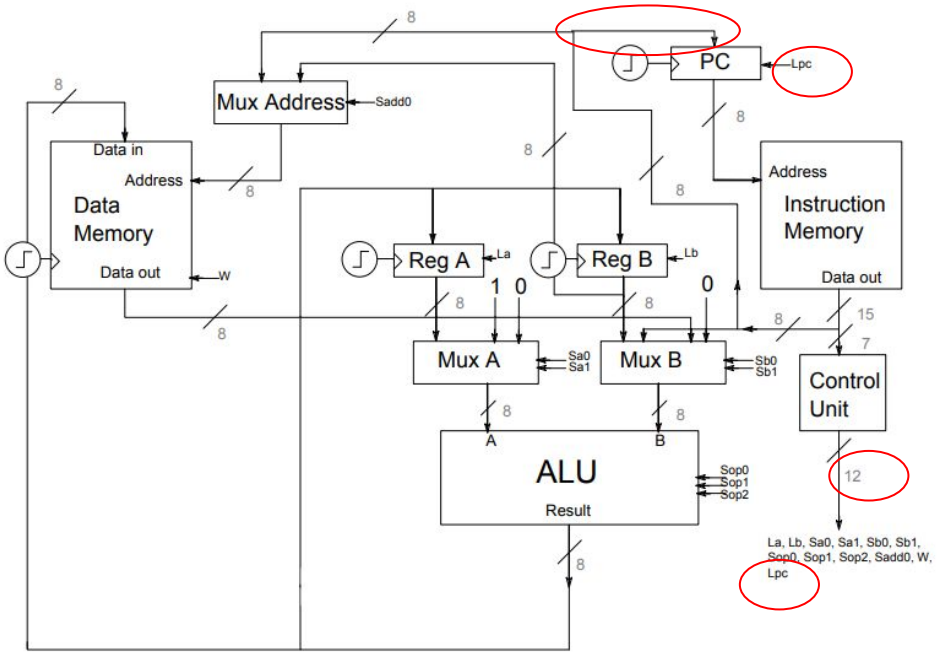


Imagen Referencia: https://github.com/IIC2343/Syllabus-anteriores/blob/main/Apuntes%20(viejos)/5%20-%20Saltos%20y%20Subrutinas.pdf

Salto condicional: Implementación

- Dado que el salto se realiza con las condiciones ocurridas en la operación anterior
- Se necesita un nuevo registro **Status**
- Su salida se conecta la Unidad de Control

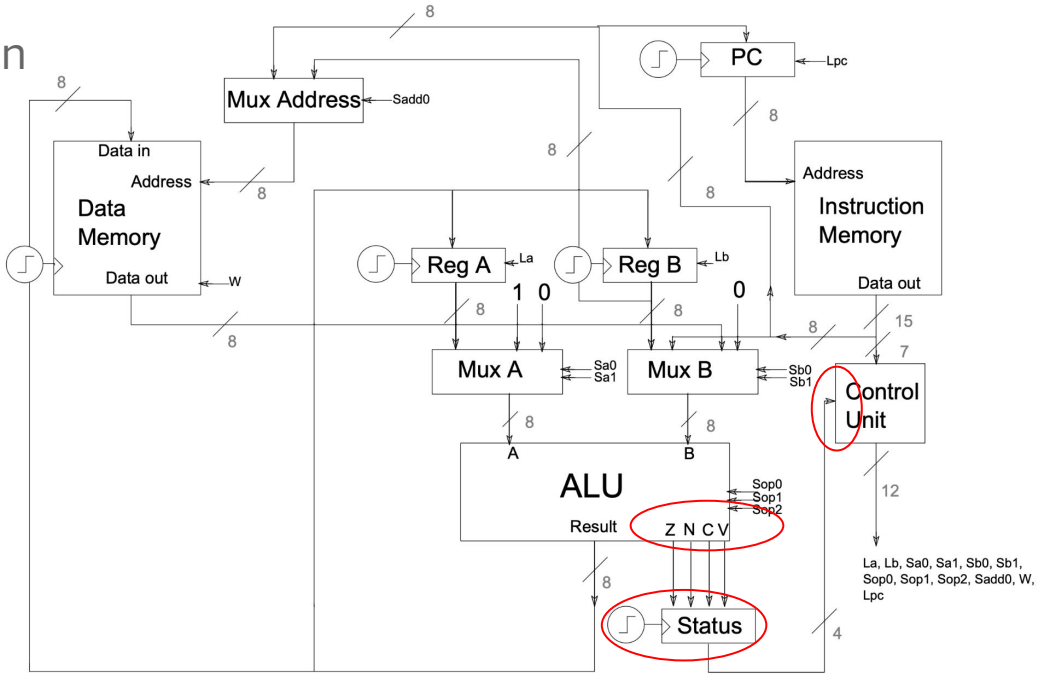


Imagen Referencia: https://github.com/IIC2343/Syllabus-anteriores/blob/main/Apuntes%20(viejos)/5%20-%20Saltos%20y%20Subrutinas.pdf

Salto Resumen

Instrucción	Operandos	Operación	Condiciones	Ejemplo de uso
CMP	A,B	A-B		
	A,Lit	A-Lit		CMP A,0
JMP	Dir	PC = Dir		JMP end
JEQ	Dir	PC = Dir	Z=1	JEQ label
JNE	Dir	PC = Dir	Z=0	JNE label
JGT	Dir	PC = Dir	N=0 y Z=0	JGT label
JLT	Dir	PC = Dir	N=1	JLT label
JGE	Dir	PC = Dir	N=0	JGE label
JLE	Dir	PC = Dir	Z=1 o N=1	JLE label
JCR	Dir	PC = Dir	C=1	JCR label
JOV	Dir	PC = Dir	V=1	JOV label

Imagen Referencia: https://github.com/IIC2343/Syllabus-anteriores/blob/main/Apuntes%20(viejos)/5%20-%20Saltos%20y%20Subrutinas.pdf

Introducción del curso:

- Un computador lo definimos como una **máquina programable que ejecuta programas.** ✓
- Para programar necesitamos:
    - Datos: números (enteros, reales) , texto, imágenes, etc ✓
    - Operaciones: suma, resta, multiplicación, división, etc ✓
    - Variables: simples, arreglos ✓
    - Control de flujo: comparaciones, manejo de ciclos ✓
- Ahora podemos volver a revisar un que es un programa...

Introducción: ¿Qué es un programa?

```
def encontrar_maximo(arreglo):
    largo_maximo = len(arreglo)

    maximo = arreglo[0]
    i = 0

    while i < largo_maximo:
        if arreglo[i] > maximo:
            maximo = arreglo[i]
        i += 1

    return maximo
```

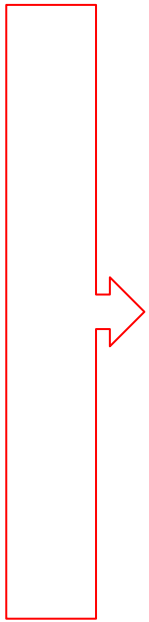
Introducción: ¿Qué es un programa?

```
def encontrar_maximo(arreglo):
    largo_maximo = len(arreglo)

    maximo = arreglo[0]
    i = 0

    while i < largo_maximo:
        if arreglo[i] > maximo:
            maximo = arreglo[i]
        i += 1

    return maximo
```



NOS FALTA!!

SUBRUTINAS

¿Dudas?

Subrutina: Ejemplo

- Con las herramientas actuales podemos escribir este código en assembly

```
def encontrar_maximo(arreglo):
    largo_maximo = len(arreglo)

    maximo = arreglo[0]
    i = 0

    while i < largo_maximo:
        if arreglo[i] > maximo:
            maximo = arreglo[i]
        i += 1

    return maximo
```

Subrutina: Ejemplo

- Definimos las variables:

```
DATA:
arr 1
    5
    3
len 3
max -1
i   0
bool 0
```

```
def encontrar_maximo(arreglo):
    largo_maximo = len(arreglo)

    maximo = arreglo[0]
    i = 0

    while i < largo_maximo:
        if arreglo[i] > maximo:
            maximo = arreglo[i]
        i += 1

    return maximo
```

Subrutina: Ejemplo

- Inicializar:

```
CODE:
update_max:
MOV A,(i)
MOV (max),A
MOV A, (bool)
CMP A, 1
JEQ increment
```

```python
def encontrar_maximo(arreglo):
    largo_maximo = len(arreglo)

    maximo = arreglo[0]
    i = 0

    while i < largo_maximo:
        if arreglo[i] > maximo:
            maximo = arreglo[i]
        i += 1

    return maximo
```

Subrutina: Ejemplo

- Iterador:

```
while:
MOV A,(i)
MOV B,(len)
CMP A,B
JLT if
JMP end
```

```python
def encontrar_maximo(arreglo):
    largo_maximo = len(arreglo)

    maximo = arreglo[0]
    i = 0

    while i < largo_maximo:
        if arreglo[i] > maximo:
            maximo = arreglo[i]
        i += 1

    return maximo
```

Subrutina: Ejemplo

- Condicional:

```
if:
MOV A, 1
MOV (bool), A
MOV B,(i)
MOV A,(B)
MOV B,(max)
CMP A,B
JGT update_max
```

```python
def encontrar_maximo(arreglo):
    largo_maximo = len(arreglo)

    maximo = arreglo[0]
    i = 0

    while i < largo_maximo:
        if arreglo[i] > maximo:
            maximo = arreglo[i]
        i += 1

    return maximo
```

Subrutina: Ejemplo

- Incrementar:

```
increment:
MOV B,(i)
INC B
MOV (i),B
JMP while
```

```
def encontrar_maximo(arreglo):
    largo_maximo = len(arreglo)

    maximo = arreglo[0]
    i = 0

    while i < largo_maximo:
        if arreglo[i] > maximo:
            maximo = arreglo[i]
        i += 1

    return maximo
```

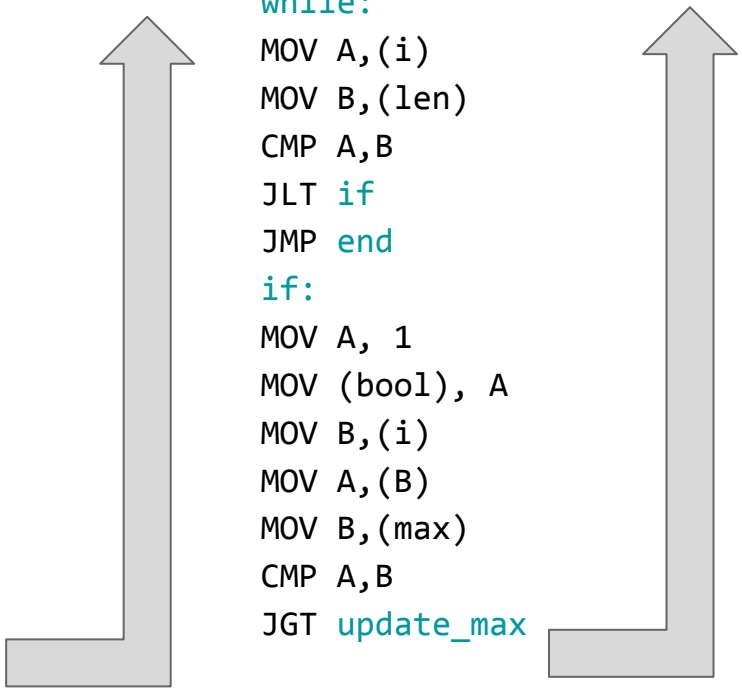
Subrutina: Ejemplo

- El código final se vería:

```
DATA:
arr 1
     5
     3
len 3
max -1
i   0
bool 0
CODE:
update_max:
MOV A,(i)
MOV (max),A
MOV A, (bool)
CMP A, 1
JEQ increment
```

```
while:
MOV A,(i)
MOV B,(len)
CMP A,B
JLT if
JMP end
if:
MOV A, 1
MOV (bool), A
MOV B,(i)
MOV A,(B)
MOV B,(max)
CMP A,B
JGT update_max
```

```
increment:
MOV B,(i)
INC B
MOV (i),B
JMP while
end:
MOV A,(max)
MOV B,(max)
JMP end
```



¿Dudas?

Subrutina: Ejemplo

- Con las herramientas actuales podemos escribir este código en assembly
- Si necesitamos calcular el máximo de 5 arreglos, significa reescribir en assembly **el mismo código múltiples veces**

```
def encontrar_maximo(arreglo):
    largo_maximo = len(arreglo)

    maximo = arreglo[0]
    i = 0

    while i < largo_maximo:
        if arreglo[i] > maximo:
            maximo = arreglo[i]
        i += 1

    return maximo
```

Subrutina: Requisitos

- Parámetro de Entrada
- Valor de retorno
- Llamada a la subrutina (salto de ida y de vuelta)

```
def encontrar_maximo(arreglo):
    largo_maximo = len(arreglo)

    maximo = arreglo[0]
    i = 0

    while i < largo_maximo:
        if arreglo[i] > maximo:
            maximo = arreglo[i]
        i += 1

    return maximo
```

Subrutina: Parámetro de Entrada y Valor Retorno

- Tenemos dos opciones:
- **Almacenamiento por Registro:** Muy limitado en la arquitectura actual
- **Almacenamiento por Memoria:** Ocupar espacio de la memoria para almacenar. El limitante es más lento pero dado el espacio es el que **utilizaremos**

Subrutina: Llamada y Retorno

- Tenemos el desafío de saber el **valor al que estamos saltamos**
- Necesitamos **Almacenar el valor del program counter**

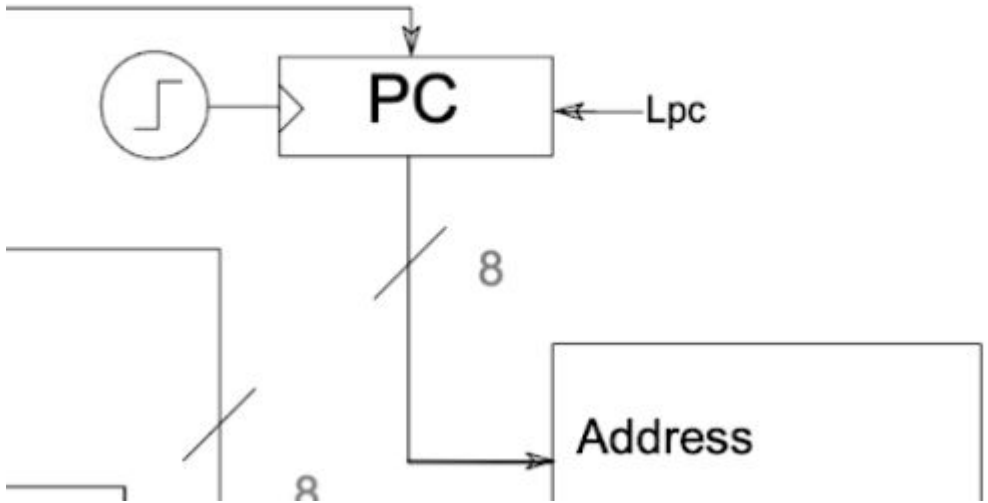


Imagen Referencia: https://github.com/IIC2343/Syllabus-anteriores/blob/main/Apuntes%20(viejos)/5%20-%20Saltos%20y%20Subrutinas.pdf

Subrutina: Implementación

- Agregamos un **Mux DataIn** a la entrada de la Memoria
- Agregamos un **Mux PC** a la entrada del Program Counter
- ¿Cómo nos aseguramos que no choquen las variables?

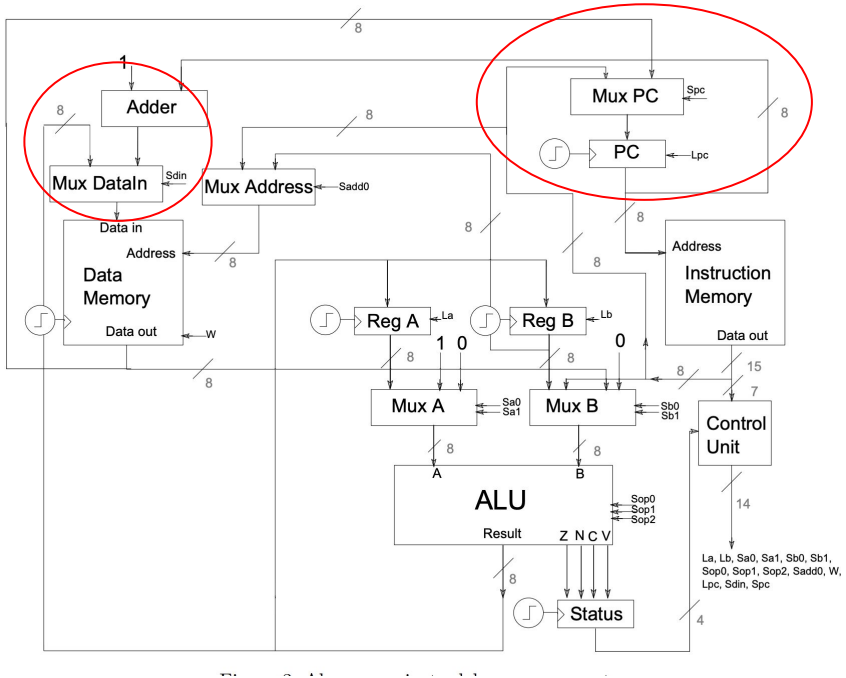


Imagen Referencia: https://github.com/IIC2343/Syllabus-anteriores/blob/main/Apuntes%20(viejos)/5%20-%20Saltos%20y%20Subrutinas.pdf

Subrutina: Implementación - SP

- Agregamos un nuevo registro **SP** a la entrada del Mux Address
- Su valor parte en 255 al ser de 8 bits (el valor más alto en base 10)

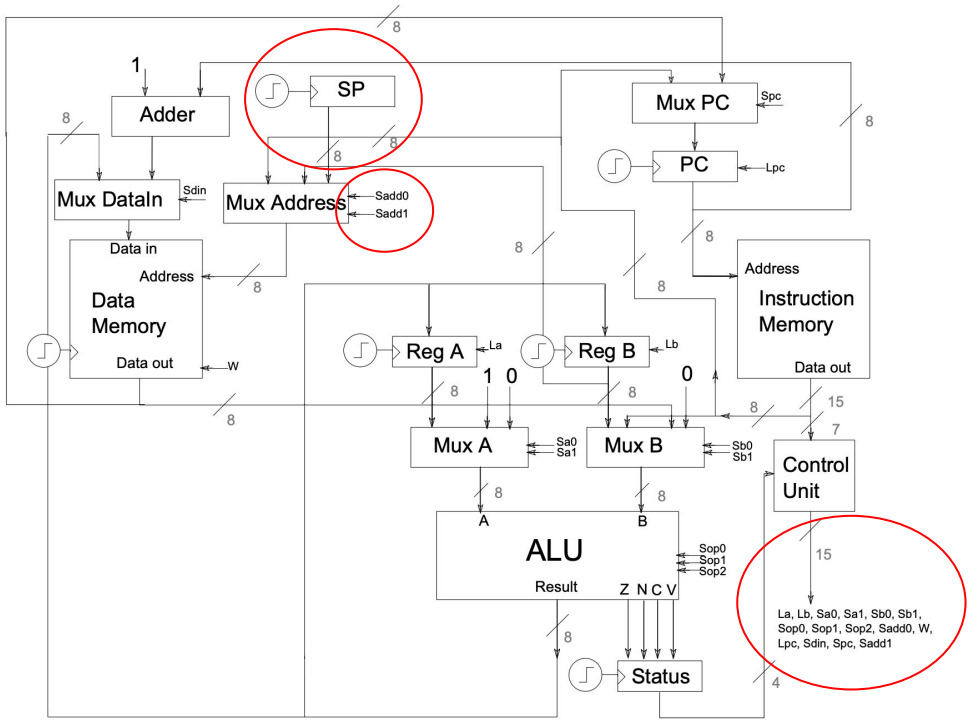


Imagen Referencia: https://github.com/IIC2343/Syllabus-anteriores/blob/main/Apuntes%20(viejos)/5%20-%20Saltos%20y%20Subrutinas.pdf

¿Dudas?

Subrutina: Memoria de Stack

- Utilizaremos el modelo de pila o en inglés Stack
- Si el *stack* choca con las variables definidas se produce un **stackoverflow**

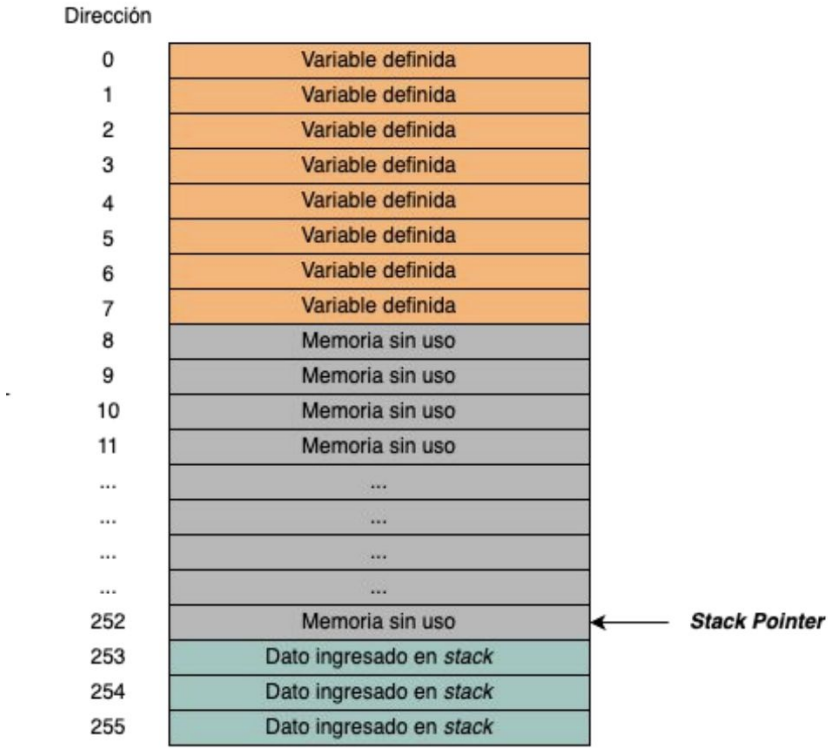


Imagen Referencia:https://github.com/IIC2343/Syllabus-2023-2/blob/main/clases/s1-s3/Clase%205%20-%20Saltos%20y%20Subrutinas.pdf

Subrutina: Assembly - CALL

- CALL label
- Almacena el valor PC+1 en la dirección de memoria SP
- Reduce en una unidad el contador SP
- Se realiza en **un ciclo de clock**

```
...
...
CALL subrutina1
...
```

$S_{DIn} = PC+1; S_{Add} = SP; W = 1;$

$Dec_{SP} = 1; S_{PC} = LIT; L_{PC} = 1$

Imagen Referencia:https://github.com/IIC2343/Syllabus-2023-2/blob/main/clases/s1-s3/Clase%205%20-%20Saltos%20y%20Subrutinas.pdf

Subrutina: Assembly - RET

- RET
- Aumenta en una unidad el contador SP
- Extrae de memoria el valor almacenado en la dirección SP
- Carga dicho valor en PC
- Se realiza en **dos ciclo de clock**

```
CALL subrutina2
...
RET
...
```

Ciclo 1: $Inc_{SP} = 1$

Ciclo 2: $S_{Add} = SP;$

$S_{PC} = DOUT; L_{PC} = 1$

Imagen Referencia:https://github.com/IIC2343/Syllabus-2023-2/blob/main/clases/s1-s3/Clase%205%20-%20Saltos%20y%20Subrutinas.pdf

Subrutina: Assembly - PUSH y POP

- **PUSH Reg**
- Almacena el valor de un registro en la dirección de memoria SP (toma un ciclo)
- **POP Reg**
- Aumenta en una unidad el contador SP y almacena en el registro Reg el dato almacenado en la dirección SP (toma dos ciclos)

```
MOV  A,5
MOV  B,3
PUSH A
PUSH B
CALL subrutina
POP  B
POP  A
```

Imagen Referencia: https://github.com/IIC2343/Syllabus-anteriores/blob/main/Apuntes%20(viejos)/5%20-%20Saltos%20y%20Subrutinas.pdf

Subrutina: Assembly - Resumen

Instrucción	Operandos	Operación	Condiciones	Ejemplo de uso
CALL	Dir	Mem[SP] = PC + 1, SP- -, PC = Dir		CALL func
RET		SP++		-
		PC = Mem[SP]		-
PUSH	A	Mem[SP] = A, SP- -		-
PUSH	B	Mem[SP] = B, SP- -		-
POP	A	SP++		-
		A = Mem[SP]		-
POP	B	SP++		-
		B = Mem[SP]		-

Imagen Referencia: https://github.com/IIC2343/Syllabus-anteriores/blob/main/Apuntes%20(viejos)/5%20-%20Saltos%20y%20Subrutinas.pdf

¿Dudas?

Introducción del curso:

- Un computador lo definimos como una **máquina programable que ejecuta programas.**
- Para programar necesitamos:
    - Datos: números (enteros, reales) , texto, imágenes, etc
    - Operaciones: suma, resta, multiplicación, división, etc
    - Variables: simples, arreglos
    - Control de flujo: comparaciones, manejo de ciclos
- Y nos permite re-utilizar trozos de código

Introducción del curso:

- Un computad[illegible]s co[illegible] **progra[illegible]e que ejecuta programas** ✔
- Para prog[illegible]
  - [illegible]
  - [illegible]
  - [illegible]
  - [illegible]
- Y nos pe[illegible]te

Imagen Referencia: https://aceattorney.fandom.com/wiki/Objection!

Introducción del curso:

- Un computador lo definimos como una **máquina programable que ejecuta programas.** ✓
- Para programar necesitamos: ?
    - Datos: números (enteros, reales) , texto, imágenes, etc ??
    - Operaciones: suma, resta, multiplicación, división, etc ✓
    - Variables: simples, arreglos ✓
    - Control de flujo: comparaciones, manejo de ciclos ✓
- Para poder tener números distinto a enteros necesitamos ¡**punto flotante**!

Clase 08 - Subrutinas

IIC2343 - Arquitectura de Computadores

Profesor:
- Felipe Valenzuela González
Correo:
frvalenzuela@alumni.uc.cl