Introducción

En esta arquitectura no tenemos soporte nativo para el manejo de pilas.
Veamos en que consiste el manejo de pila del MSX88:

- Tiene un registro SP, que se usa solo para el manejo de la pila y que inicialmente vale 08000h.
- Tenemos una instrucción PUSH que nos permite *apilar* valores de 16 bits en la misma.
- Tenemos una instrucción POP que nos permite *desapilar* valores de 16 bits de la misma.
- La pila también se utiliza para el manejo de subrutinas, pero ya tenemos un reemplazo para eso.

Pila

- El MSX88 posee un registro especifico para la Pila
- WinMIPS64 posee 32 registros de proposito general, podemos elegir uno y usarlo explusivamente para la pila.
- La convención de nombres de registros ya vistos ya contempla al registro *r29* como el registro *$sp*.
- El registro SP esta inicializado en 08000h, debemos inicializar el registro de pila a un valor adecuado.
- Podemos inicializar el registro *$sp* en 0400h, que es la posición de memoria de datos más alta que tenemos en WinMIPS64.

```
.code
daddi $sp, $0, 0x400
...
```

Apilar

La instrucción PUSH del MSX88 es la encargada de apilar un valor en la pila. Veamos en que consiste apilar un valor:

- $(SP) \leftarrow (SP) - 2$; *Decremento Puntero de pila*
- $[SP + 1 : SP] \leftarrow (fuente)$; *Guardo valor en la pila*

Para simular esta operación deberiamos:

- Decrementar en 8 el puntero de pila. ¿Por qué 8?
- Guardar en la posición de memoria del puntero de pila el valor.

Apilar - Detalle

Apilando el valor del registro $t1

```
...
daddi $sp, $sp, -8
sd $t1, 0($sp)          ; Guardo en la pila
...
```

Desapilar

La instrucción POP del MSX88 es la encargada de desapilar un valor de la pila. Veamos en que consiste:

- $(\textit{fuente}) \leftarrow [SP + 1 : SP]$ *; Guardo valor de la pila*
- $(SP) \leftarrow (SP) + 2$ *; Incremento el Puntero de pila*

Para simular esta operación deberiamos:

- Leer el dato de la posición de memoria del puntero de pila y guardarlo en un registro.
- Incrementar en 8 el puntero de pila.

Desapilar - Detalle

Desapilando en el registro $s1

```
...
ld $s1, 0($sp)          ; Leo desde la pila
daddi $sp, $sp, +8
...
```

Multiples valores

Veamos que pasa si se apilan varios valores seguidos:

```
...
daddi $sp, $sp, -8
sd $s1, 0($sp)        ; Apilo s1

daddi $sp, $sp, -8
sd $s2, 0($sp)        ; Apilo s2

daddi $sp, $sp, -8
sd $s3, 0($sp)        ; Apilo s3

daddi $sp, $sp, -8
sd $s4, 0($sp)        ; Apilo s4

...
```

Podriamos agrupar los *daddi* y ver si podemos tener menos instrucciones.

Multiples valores

```
...
daddi $sp, $sp, -32 ; Reservo 32 bytes
sd $s1, 24($sp) ; Apilo s1
sd $s2, 16($sp) ; Apilo s2
sd $s3, 8($sp)  ; Apilo s3
sd $s4, 0($sp)  ; Apilo s4
...
```

- Ajustamos *$sp* al inicio, le restamos 8 por cada registro a apilar.
- Luego guardamos cada registro, desplazandolo de a 8 posiciones.

Multiples valores - Continuación

Y para desapilar, lo hacemos de manera similar

```
...

ld $s1, 24($sp) ; Restauro s1
ld $s2, 16($sp) ; Restauro s2
ld $s3, 8($sp)  ; Restauro s3
ld $s4, 0($sp)  ; Restauro s4
daddi $sp, $sp, +32
...
```

Pilas y Subrutina

Toda subrutina se dividirá siempre en tres partes

- *Prólogo*: se resguarda en la pila todos los registros que deban ser preservados
- *Cuerpo* de la subrutina: con el código propio de la misma
- *Epílogo*: se restauran los registros preservados en el prólogo

```
subrutina:
    daddi $sp, $sp, -16
    sd $ra, 8($sp) ; Apilo ra
    sd $s0, 0($sp)  ; Apilo s0

    ...

    ld $ra, 8($sp)  ; Restauro ra
    ld $s0, 0($sp)  ; Restauro s0
    daddi $sp, $sp, +16
    jr $ra  ; $ra tiene valor de retorno
```

- Si no se modifican los registros *$s0* a *$s7*, no es necesario guardarlos
- Una subrutina sencilla podria tener un prólogo y epílogo vacio

Anidamiento de Subrutinas

- Si una subrutina llama a otra subrutina, va a alterar el valor del registro *$ra*. Por lo tanto debemos preservar el valor que recibimos de *$ra*.
- Si no llama a otra subrutina, no es necesario que guarde el valor contenido en *$ra*.
- Esto es valido tanto para subrutinas que se llamen a si mismas (recursivas) como para subrutinas que llamen a otras.

Ejemplo

```
.data
valor: .word 10
result: .word 0
.text
    daddi $sp, $0, 0x400 ; Inicializa $sp
    ld $a0, valor($0)
    jal factorial
    sd $v0, result($0)
    halt
factorial: daddi $sp, $sp, -16
           sd $ra, 0($sp)
           sd $s0, 8($sp)
           beqz $a0, fin_rec
           dadd $s0, $0, $a0
           daddi $a0, $a0, -1
           jal factorial
           dmul $v0, $v0, $s0
           j fin
fin_rec:   daddi $v0, $0, 1
fin:       ld $s0, 8($sp)
           ld $ra, 0($sp)
           daddi $sp, $sp, 16
           jr $ra
```