Hardware

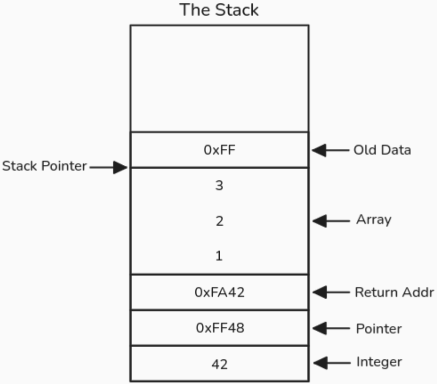
The Stack

Jean-Malo Meichel

September 2024

General Case

The Stack



The stack is:

- An in memory reserved zone
- A LIFO
- Managed by a pointer: the stack pointer
- A temporary storage by nature

The Context

The context is the **state** of the CPU at a given time.

The Context

The context is the **state** of the CPU at a given time.

CPU Registers are **limited**. Being able to reuse them is **essential**.

The Context

The context is the **state** of the CPU at a given time.

CPU Registers are **limited**. Being able to reuse them is **essential**.

A **generic routine** has no clue about already used registers.

The Context

The context is the **state** of the CPU at a given time.

CPU Registers are **limited**. Being able to reuse them is **essential**.

A **generic routine** has no clue about already used registers.

Solution: **save the context** then **restore it**.

The Context

The context is the **state** of the CPU at a given time.

CPU Registers are **limited**. Being able to reuse them is **essential**.

A **generic routine** has no clue about already used registers.

Solution: **save the context** then **restore it**.

The **stack** can be used for that.

M68000 Case

The M68000 Stack

In M68000 Assembly:

- A7 stores the **Stack Pointer**.

The M68000 Stack

In M68000 Assembly:

- A7 stores the **Stack Pointer**.
- The stack grows **downward** from the end of the memory.

The M68000 Stack

In M68000 Assembly:

- A7 stores the **Stack Pointer**.
- The stack grows **downward** from the end of the memory.
- A7 is used internally by instructions (jsr, rts, etc...).

The M68000 Stack

In M68000 Assembly:

- A7 stores the **Stack Pointer**.
- The stack grows **downward** from the end of the memory.
- A7 is used internally by instructions (jsr, rts, etc...).

The stack can be accessed by user to store data.

The M68000 Stack

In M68000 Assembly:

- A7 stores the **Stack Pointer**.
- The stack grows **downward** from the end of the memory.
- A7 is used internally by instructions (jsr, rts, etc...).

The stack can be accessed by user to store data.

It is generally used to store the **context**.

The M68000 Stack

In M68000 Assembly:

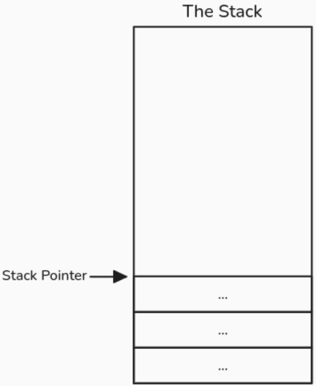
- A7 stores the **Stack Pointer**.
- The stack grows **downward** from the end of the memory.
- A7 is used internally by instructions (jsr, rts, etc...).

The stack can be accessed by user to store data.

It is generally used to store the **context**.

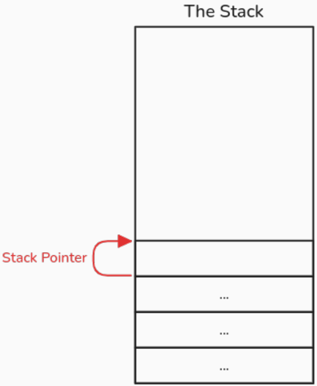
It is also involved in interrupt handling (not discussed in this lesson).

Push / Pop



Pushing to the stack:

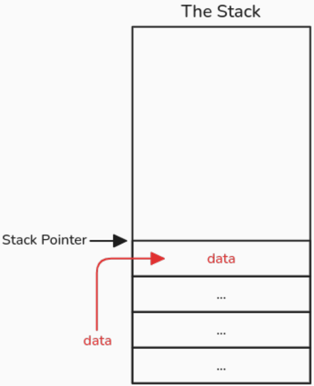
Push / Pop



Pushing to the stack:

```
suba.l      #2,A7           ; Decrement stack pointer
```

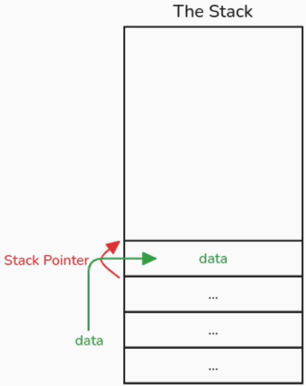
Push / Pop



Pushing to the stack:

```
suba.l      #2,A7           ; Decrement stack pointer
move.w      #data,(A7)      ; Move data to the stack
```

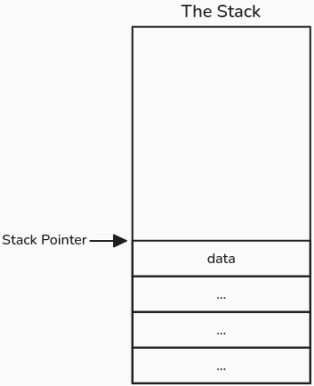
Push / Pop



Pushing to the stack:

```
move.w   #data,-(A7) ; Decrement SP then move data
```

Push / Pop

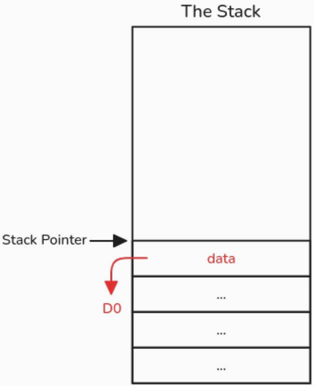


Pushing to the stack:

```
move.w   #data,-(A7) ; Decrement SP then move data
```

Popping from the stack:

Push / Pop



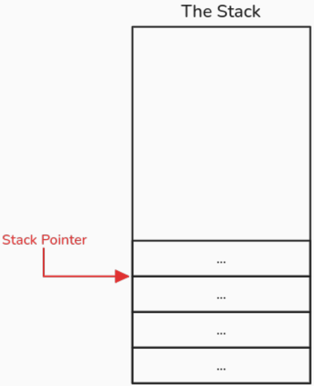
Pushing to the stack:

```
move.w   #data,-(A7) ; Decrement SP then move data
```

Popping from the stack:

```
move.w      (A7),D0    ; Retrieve data
```

Push / Pop



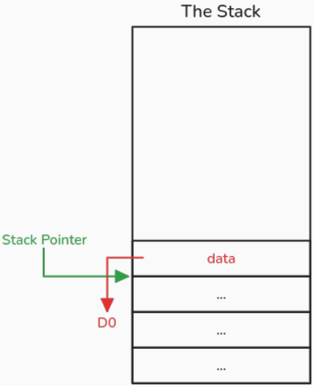
Pushing to the stack:

```
move.w   #data,-(A7) ; Decrement SP then move data
```

Popping from the stack:

```
move.w      (A7),D0    ; Retrieve data
adda.l      #2,A7     ; Increment stack pointer
```

Push / Pop



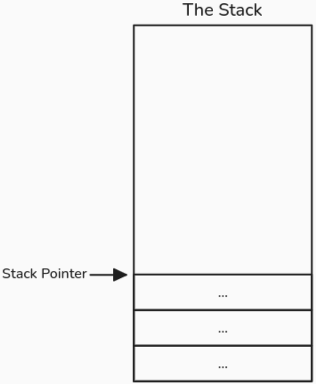
Pushing to the stack:

```
move.w   #data,-(A7) ; Decrement SP then move data
```

Popping from the stack:

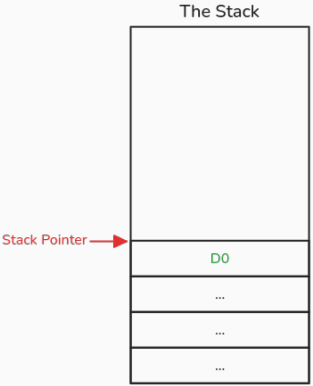
```
move.w      (A7)+,D0    ; Retrieve + Increment SP
```

Context Switch



Saving context:

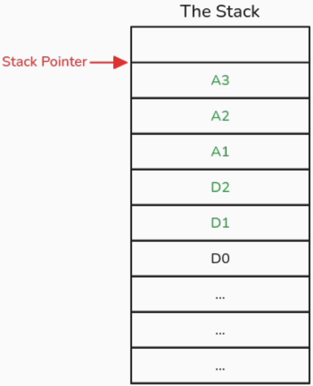
Context Switch



Saving context:

```
move.l    D0,-(A7)                ; Single register
```

Context Switch

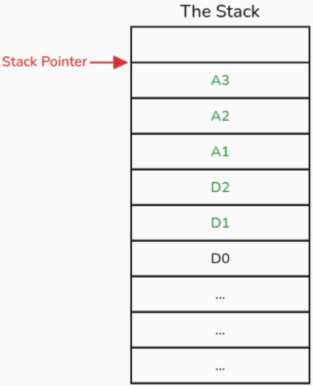


Saving context:

```
move.l   D0,-(A7)              ; Single register
movem.l  D1/D2/A1-A3,-(A7)     ; Multiple registers
```

- movem is equivalent to multiple move

Context Switch

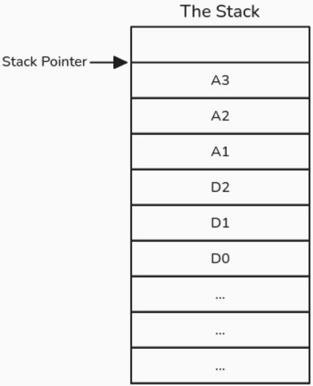


Saving context:

```
move.l   D0,-(A7)              ; Single register
movem.l  D1/D2/A1-A3,-(A7)     ; Multiple registers
```

- movem is equivalent to multiple move
- move order is managed automagically (by the assembler)

Context Switch



The Stack

Stack Pointer

A3

A2

A1

D2

D1

D0

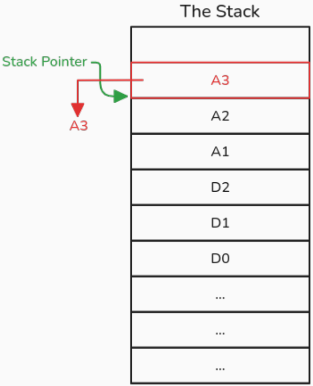
...

...

...

Retreiving context:

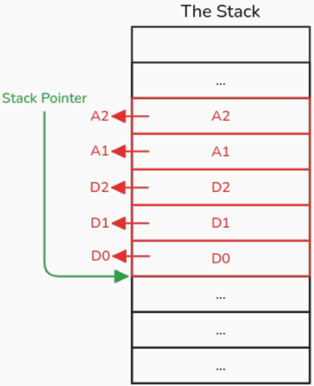
Context Switch



Retreiving context:

```
move.l      (A7)+,A3            ; Single retrieve
```

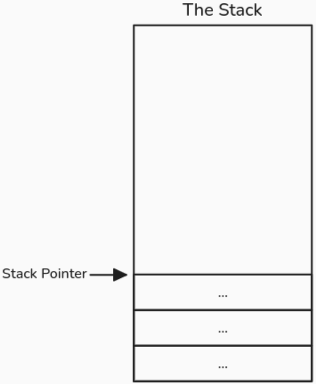
Context Switch



Retreiving context:

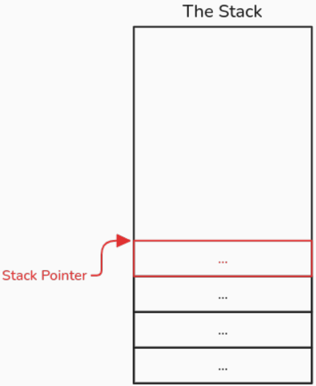
```
move.l      (A7)+,A3           ; Single retrieve
movem.l     (A7)+,D0-D2/A1/A2  ; Multiple retrieve
```

Routines



Function calls are done with the **jsr** and **rts** instructions.

Routines

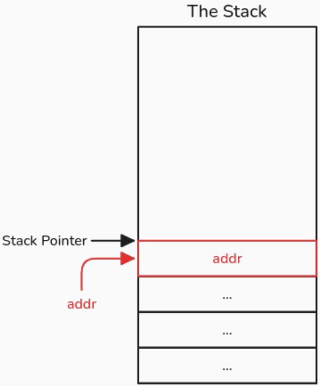


Function calls are done with the **jsr** and **rts** instructions.

jsr jumps to a label's address, it is equivalent to:

```
move.l      PC,-(A7) ; Save the return address
```

Routines

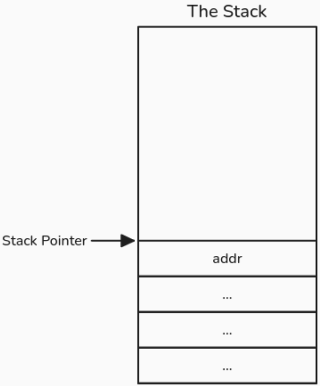


Function calls are done with the **jsr** and **rts** instructions.

jsr jumps to a label's address, it is equivalent to:

```
move.l      PC,-(A7) ; Save the return address
move.l      #addr,PC ; Jump to label
```

Routines



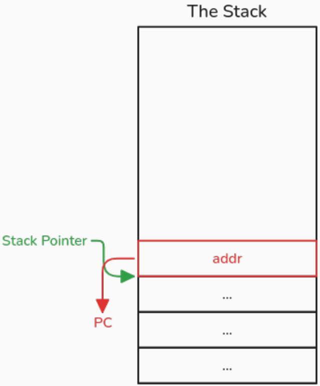
Function calls are done with the **jsr** and **rts** instructions.

jsr jumps to a label's address, it is equivalent to:

```
move.l      PC,-(A7) ; Save the return address
move.l      #addr,PC ; Jump to label
```

rts jumps back to the caller's location. it is equivalent to:

Routines



Function calls are done with the **jsr** and **rts** instructions.

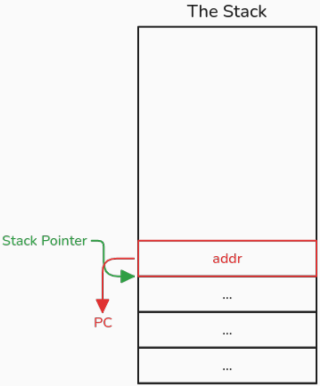
jsr jumps to a label's address, it is equivalent to:

```
move.l      PC,-(A7)  ; Save the return address
move.l      #addr,PC  ; Jump to label
```

rts jumps back to the caller's location. it is equivalent to:

```
move.l      (A7)+,PC  ; Retrieve return address
```

Routines



Function calls are done with the **jsr** and **rts** instructions.

jsr jumps to a label's address, it is equivalent to:

```
move.l      PC,-(A7) ; Save the return address
move.l      #addr,PC ; Jump to label
```

rts jumps back to the caller's location. it is equivalent to:

```
move.l      (A7)+,PC ; Retrieve return address
add.l       #2,PC    ; Jump to next instruction (
    variable size)
```

Routines

rts instruction expects the return address to be at the top of the stack.

It is **your** responsibility to ensure it is the case.

Any data pushed to the stack during a routine **must** be poped **before** the **rts** instruction.

Standard Routine Layout

Here is the look your routines may have during the practicals:

```
label                           ; Routine's label
    movem.l  D0/D1/A0-A2,-(A7) ; Save previous context

    ; ------------------ ;
    ; - Routine's body - ;
    ; ------------------ ;

    movem.l  (A7)+,D0/D1/A0-A2 ; Retrieve context
    rts                         ; Return to caller
```

Stack Alignment

All M68000 **word** size memory access **must** be aligned to words.

Stack Alignment

All M68000 **word** size memory access **must** be aligned to words.

This is a limitation **imposed** by the architecture.

Stack Alignment

All M68000 **word** size memory access **must** be aligned to words.

This is a limitation **imposed** by the architecture.

To ensure that next stack pushes **are** aligned, all byte pushes must move A7 by **two** bytes.

Stack Alignment

All M68000 **word** size memory access **must** be aligned to words.

This is a limitation **imposed** by the architecture.

To ensure that next stack pushes **are** aligned, all byte pushes must move A7 by **two** bytes.

This is **automatic** if you are using **pre-decrementation** addressing mode.

Stack Alignment

All M68000 **word** size memory access **must** be aligned to words.

This is a limitation **imposed** by the architecture.

To ensure that next stack pushes **are** aligned, all byte pushes must move A7 by **two** bytes.

This is **automatic** if you are using **pre-decrementation** addressing mode.

Misaligned stack **will** cause the CPU to **raise** an interrupt and will **crash** the simulator.

Advanced: Stack Frame

Sometimes 8 registers are not enough.

Advanced: Stack Frame

Sometimes 8 registers are not enough.

Excess variables can be stored on the stack.

Advanced: Stack Frame

Sometimes 8 registers are not enough.

Excess variables can be stored on the stack.

Usually, a **stack frame** is used for that.

Advanced: Stack Frame

A **stack frame** is a pre-reserved space in the stack.

Advanced: Stack Frame

A **stack frame** is a pre-reserved space in the stack.

It is usually reserved at the **start** of a routine.

Advanced: Stack Frame

A **stack frame** is a pre-reserved space in the stack.

It is usually reserved at the **start** of a routine.

It is then released before the routine's end.

Advanced: Stack Frame

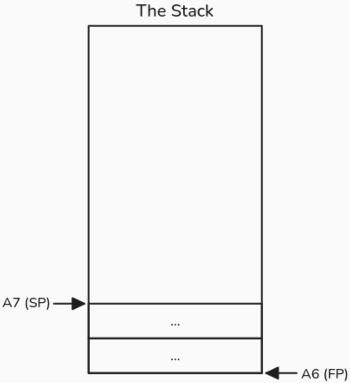
A **stack frame** is a pre-reserved space in the stack.

It is usually reserved at the **start** of a routine.

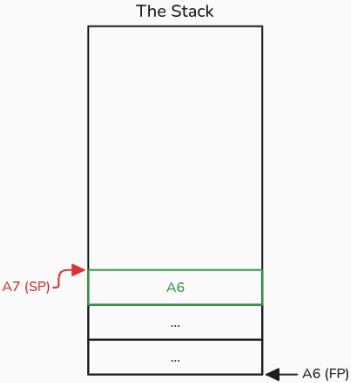
It is then released before the routine's end.

By convention, **A6** is used as the frame pointer.

Advanced: Stack Frame Example

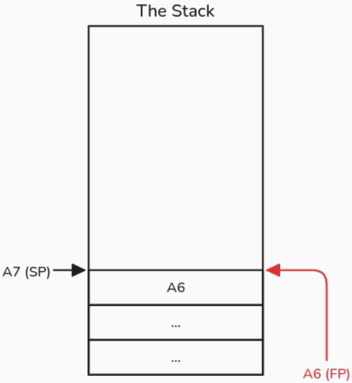


Advanced: Stack Frame Example



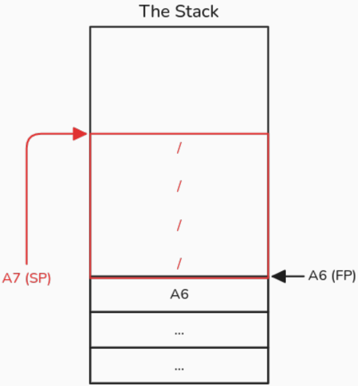
```
move.l  A6,-(A7) ; Push previous frame pointer
```

Advanced: Stack Frame Example



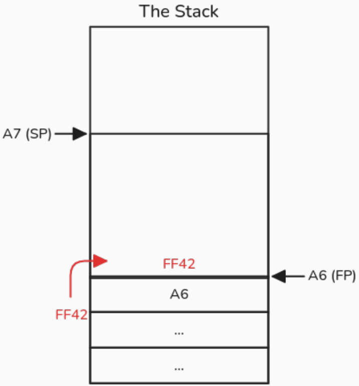
```
move.l   A6,-(A7)  ; Push previous frame pointer
movea.l  A7,A6     ; Get new frame origin
```

Advanced: Stack Frame Example



```
move.l   A6,-(A7)  ; Push previous frame pointer
movea.l  A7,A6     ; Get new frame origin
suba.l   #8,A7     ; Reserve 8 bytes
```

Advanced: Stack Frame Example

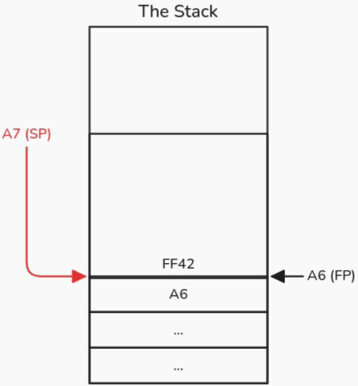


```
move.l   A6,-(A7) ; Push previous frame pointer
movea.l  A7,A6    ; Get new frame origin
suba.l   #8,A7    ; Reserve 8 bytes

move.w   #$FF42,-2(A6) ; Store something

; - Routine's body - ;
```

Advanced: Stack Frame Example



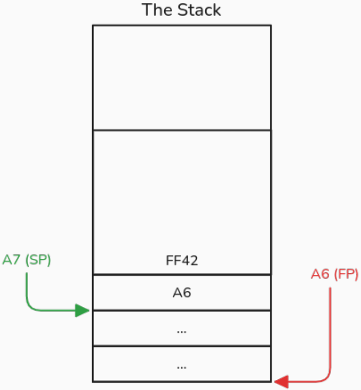
```
move.l   A6,-(A7) ; Push previous frame pointer
movea.l  A7,A6    ; Get new frame origin
suba.l   #8,A7    ; Reserve 8 bytes

move.w   #$FF42,-2(A6) ; Store something

; - Routine's body - ;

movea.l  A6,A7    ; Remove stack frame
```

Advanced: Stack Frame Example



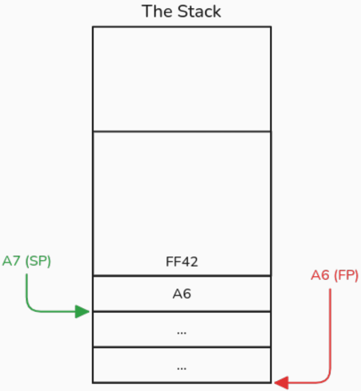
```
move.l   A6,-(A7)  ; Push previous frame pointer
movea.l  A7,A6     ; Get new frame origin
suba.l   #8,A7     ; Reserve 8 bytes

move.w   #$FF42,-2(A6) ; Store something

; - Routine's body - ;

movea.l  A6,A7     ; Remove stack frame
movea.l  (A7)+,A6  ; Restore old frame pointer
```

Advanced: Stack Frame Example



```
link     A6,#-8     ; Create stack frame

suba.l   #8,A7      ; Reserve 8 bytes

move.w   #$FF42,-2(A6) ; Store something

; - Routine's body - ;

unlk     A6         ; Remove stack frame
```