Stack

R. Ferrero

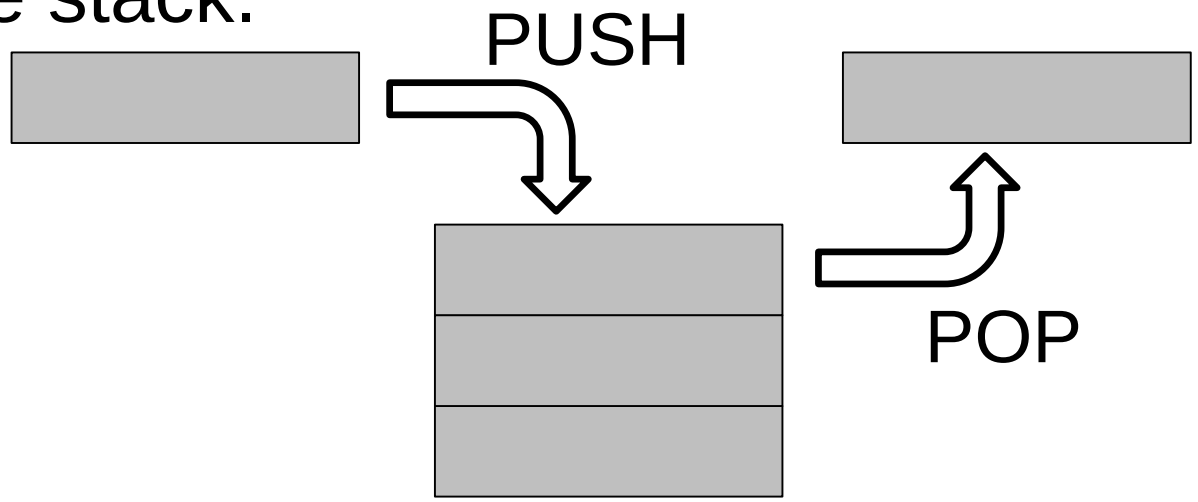
Politecnico di Torino

Dipartimento di Automatica e Informatica (DAUIN)

Torino - Italy

Stack

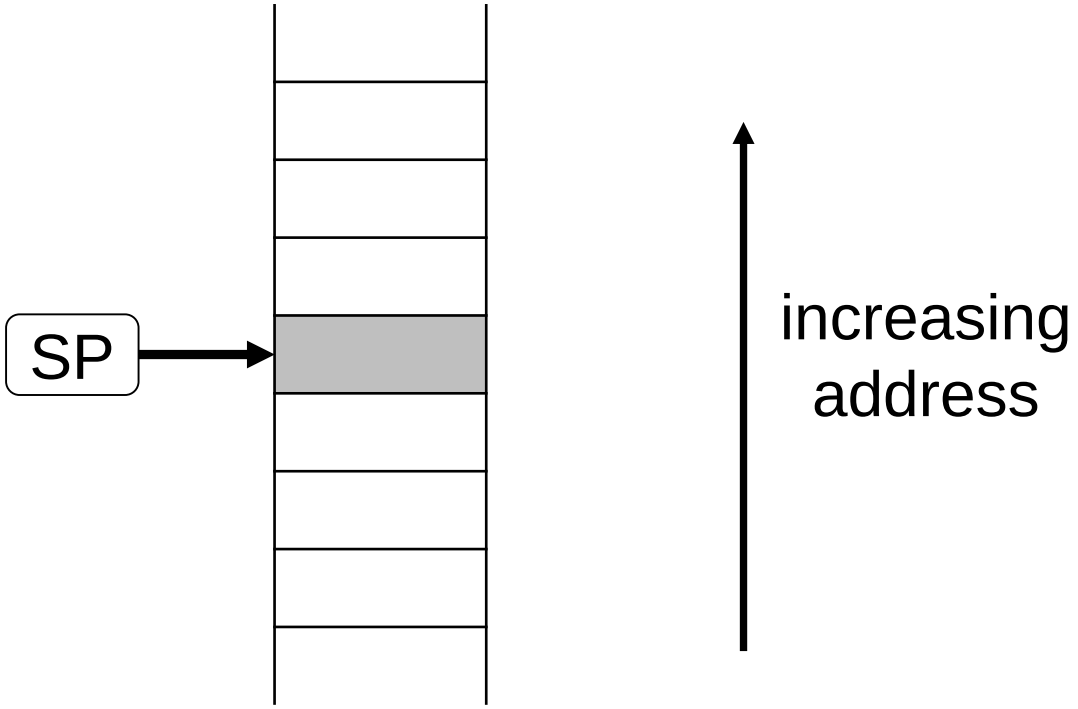
- A stack is a Last In-First Out (LIFO) queue.
- Data is pushed (written) to and popped (read) from the top of the stack.
- The stack pointer contains the address of the top of the stack.



Types of stack

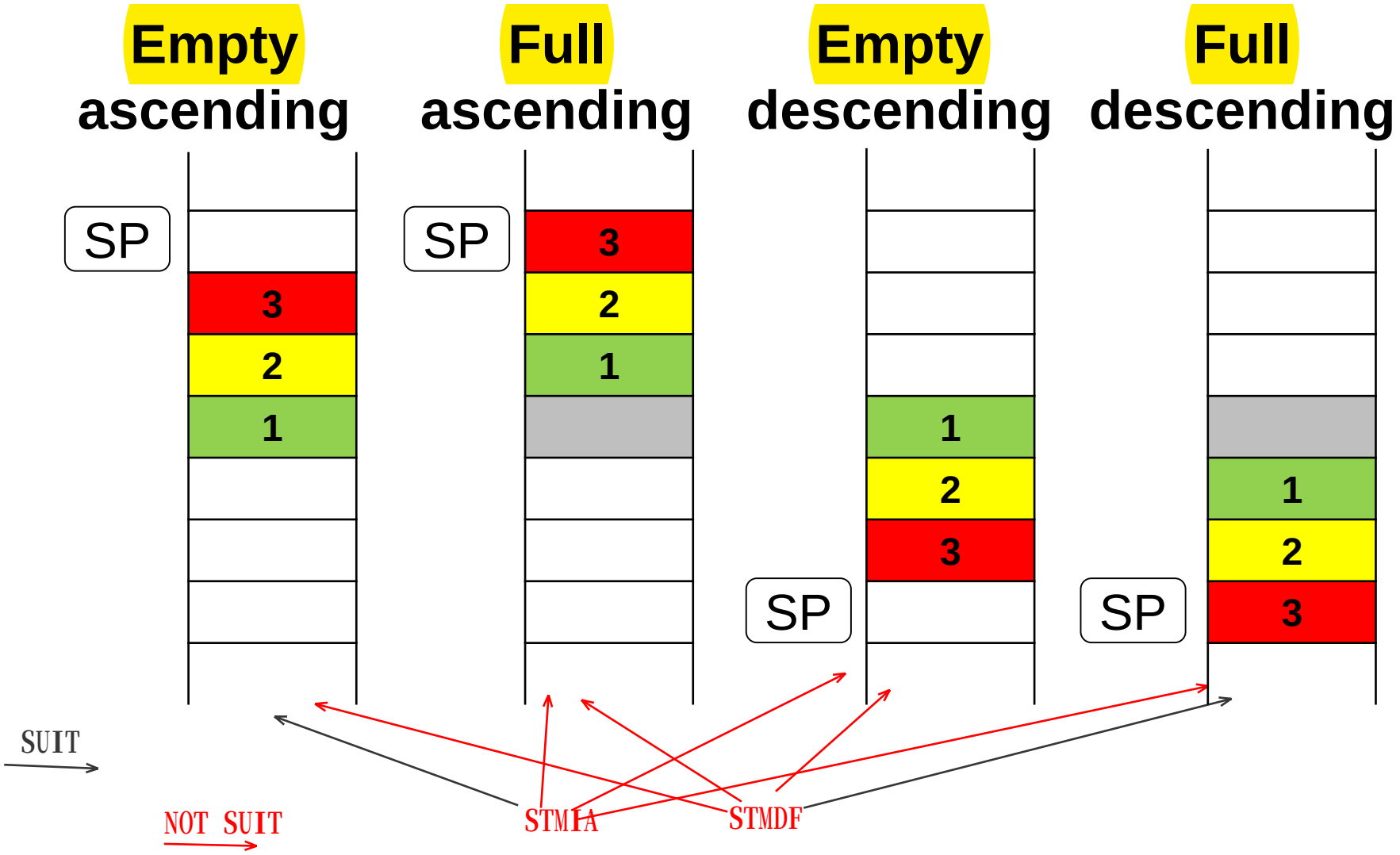
- Stack pointer updating:
  - *descending stack*: the address of the stack pointer decreases after a push push from 100(top) to 0(bottom)
  - *ascending stack*: the address of the stack pointer increases after a push push from 0(bottom) to 100(top)
- Entry pointed by the stack pointer:
  - *empty stack*: the stack pointer points to the entry where new data will be pushed
  - *full stack*: the stack pointer points to the last pushed entry.

Example with PUSH: initial state

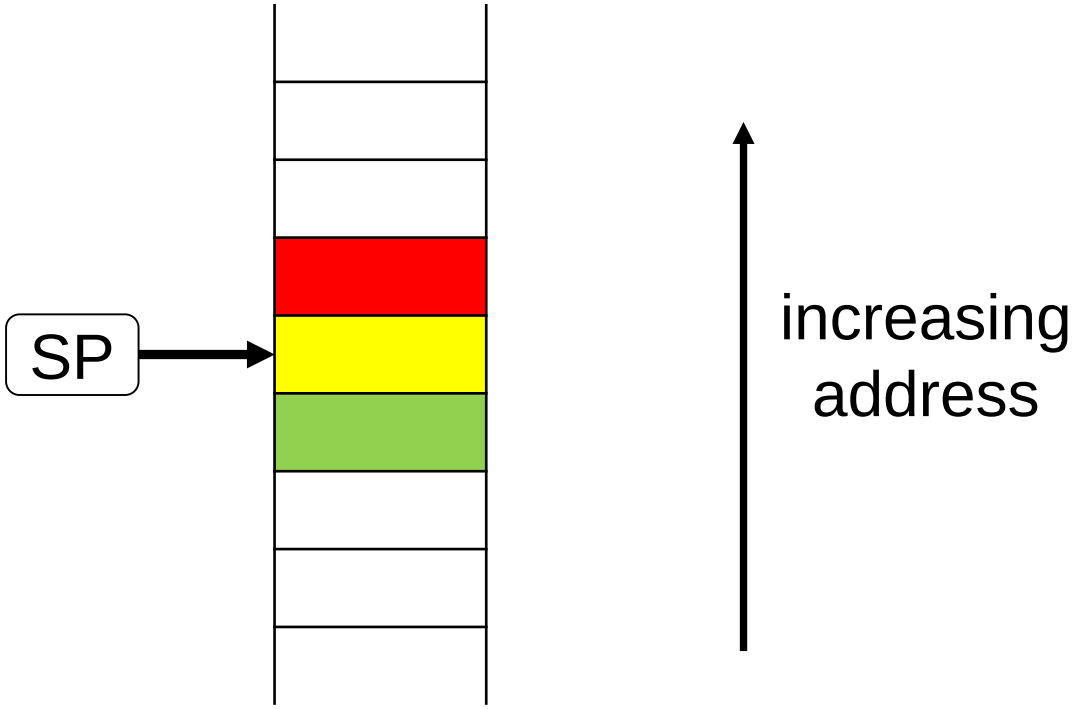


- 3 PUSH: 1 2 3

Example after 3 PUSH

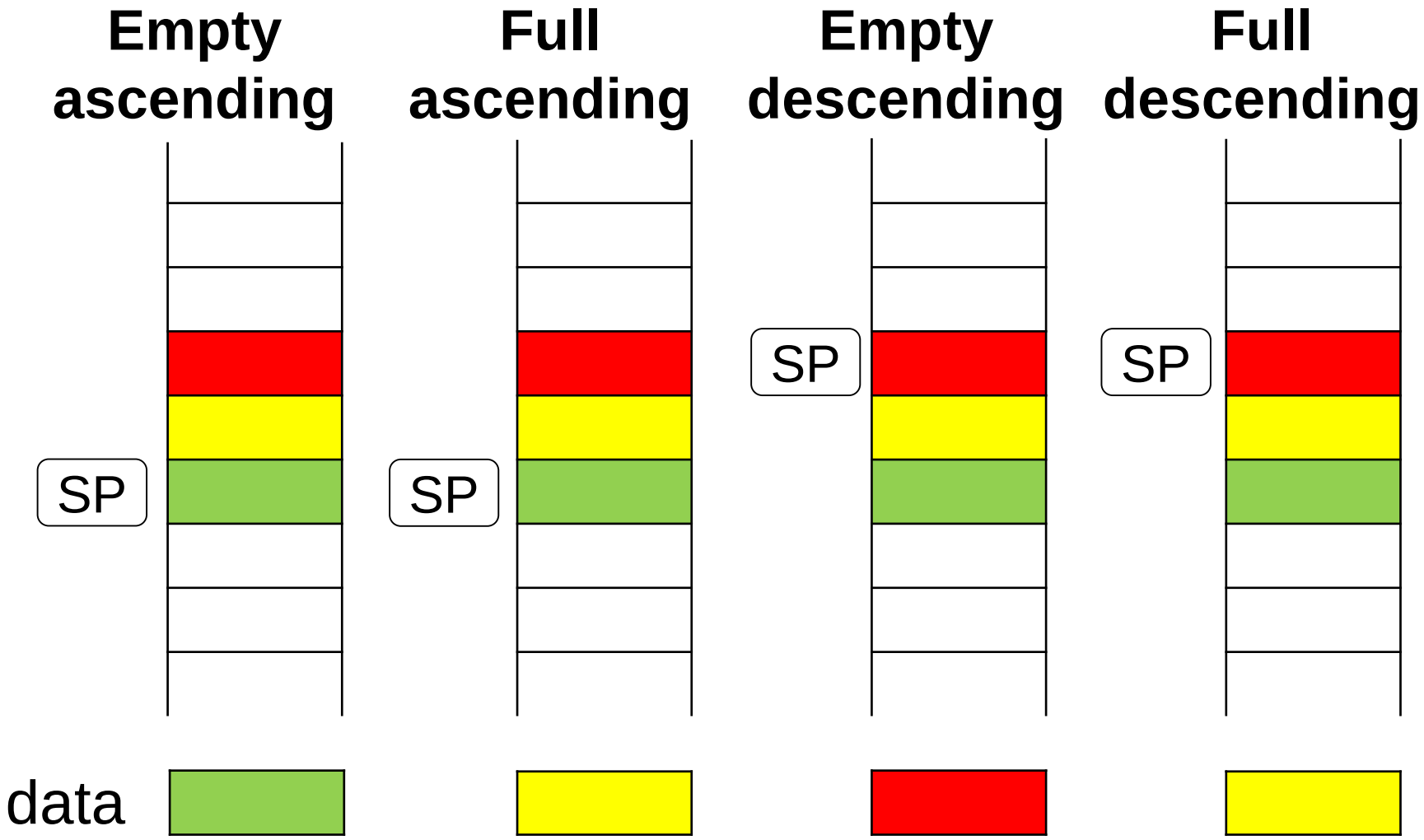


Example with POP: initial state



- 1 POP

Example after 1 POP



LDM and STM

- They transfer one or more words:

  `LDM{xx}/STM{xx} <Rn>{!}, <regList>`

- `Rn` is the base register
- `xx` specifies the addressing mode, i.e., how and when `Rn` is updated during the instruction
- at the end of the instruction:
  - with `!`, `Rn` is set to the updated value
  - without `!`, `Rn` is set to the initial value
- `regList` is a list of registers.

List of registers

- Consecutive registers are indicated by separating the initial and final registers with a dash
- Non consecutive registers are separated with a comma.
- Example: `{r0-r4, r10, LR}` indicates `r0`, `r1`, `r2`, `r3`, `r4`, `r10`, `r14`.
- `SP` can not appear in the list.
- `PC` can appear only with `LDM` and only if `LR` is missing in the list.

Order of registers in the list

- The order of registers does not matter.
- Registers are automatically sorted in increasing order:
  - the lowest register is stored into / loaded from the lowest memory address
  - the highest register is stored into / loaded from the highest memory address
- Example: `{r8, r1, r3-r5, r14}` indicates `r1`, `r3`, `r4`, `r5`, `r8`, `r14`.

Addressing modes

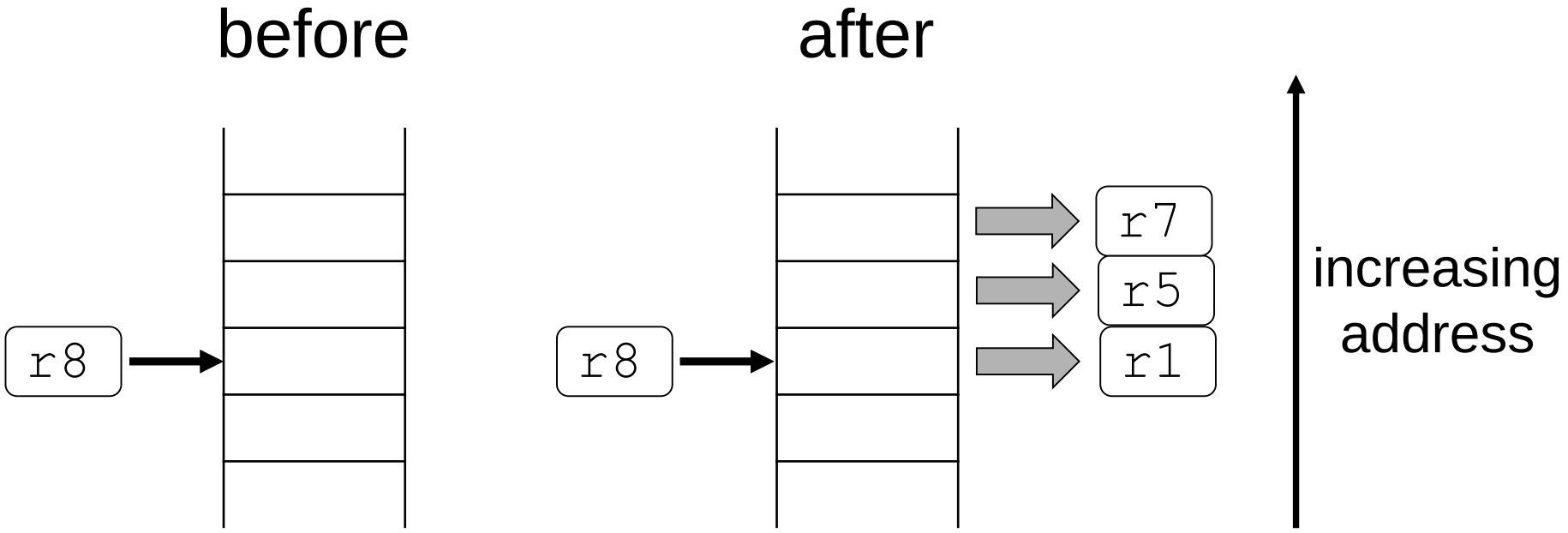
- `IA`: increment after (default)
  1. memory is accessed at the address specified in the base register
  2. base register is incremented by 1 word (4 bytes)
  3. if there are other registers in the list, go to 1.
- `DB`: decrement before
  1. base register is decremented by 1 word (4 bytes)
  2. memory is accessed at the address specified in the base register
  3. if there are other registers in the list, go to 1.

LDMIA: an example

lord the values behind address r8 one by one

and does't update r8

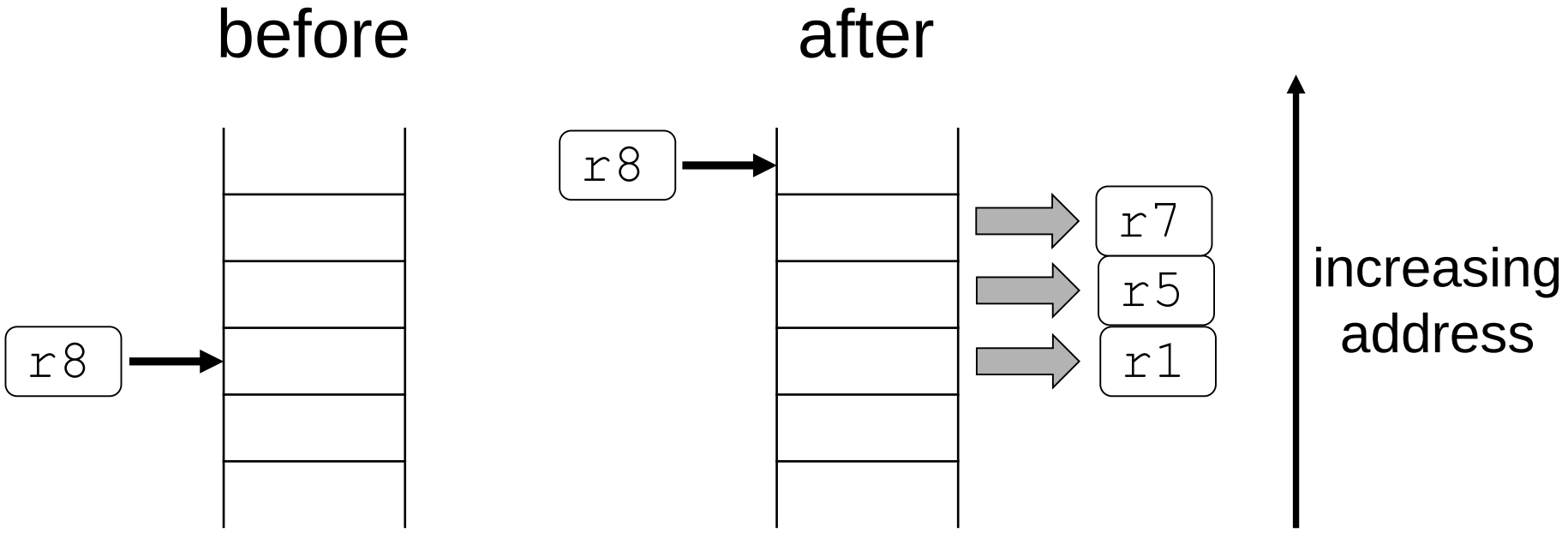
```
LDMIA r8, {r1, r5, r7}
LDM r8, {r1, r5, r7}
```



LDMIA with '!': an example

update r8

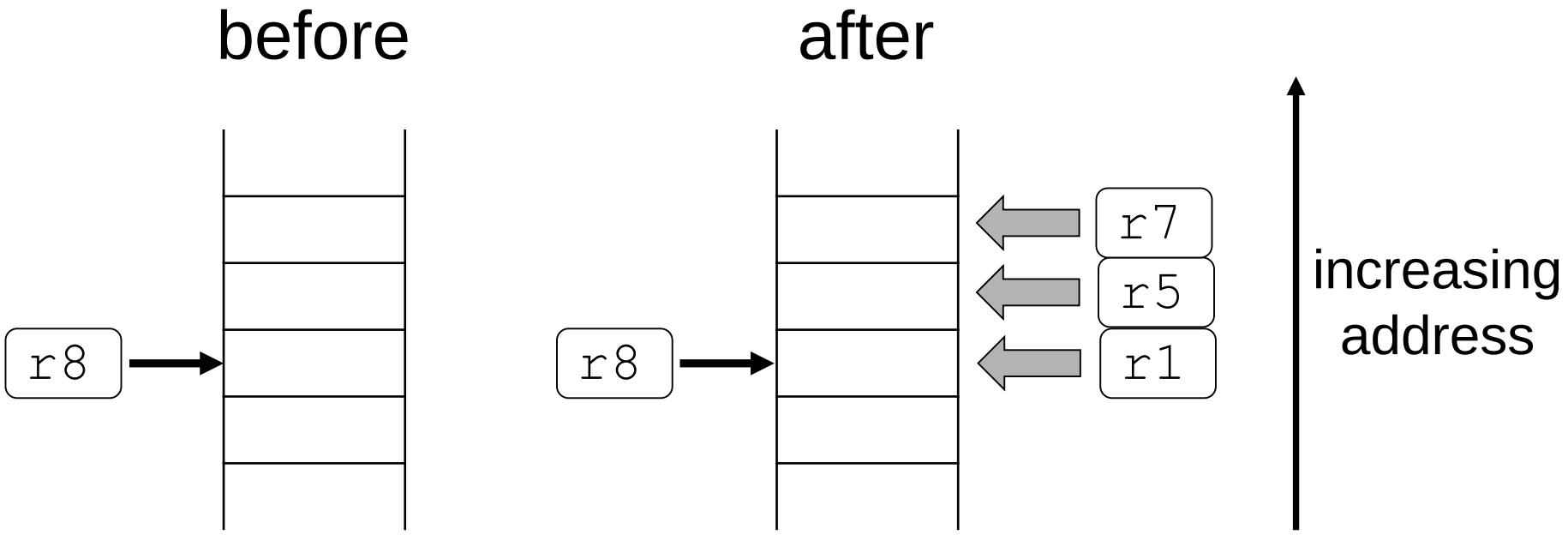
```
LDMIA r8!, {r1, r5, r7}
LDM r8!, {r1, r5, r7}
```



STMIA: an example

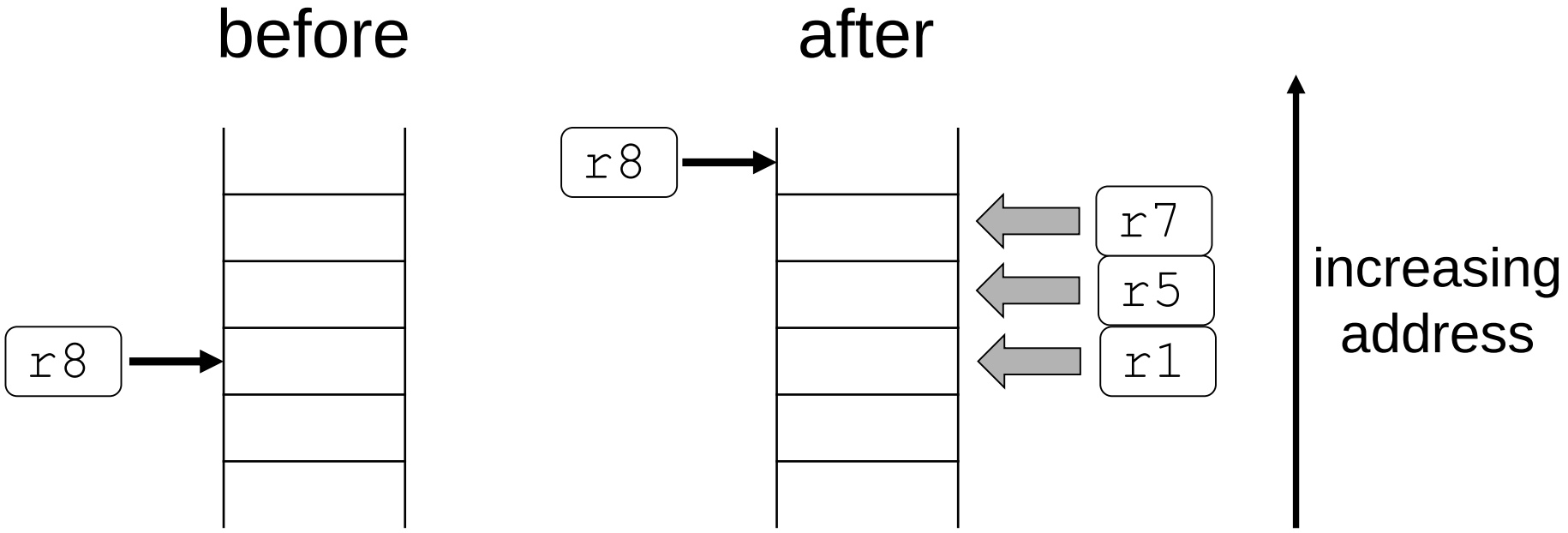
store

```
STMIA r8, {r1, r5, r7}
STM r8, {r1, r5, r7}
```



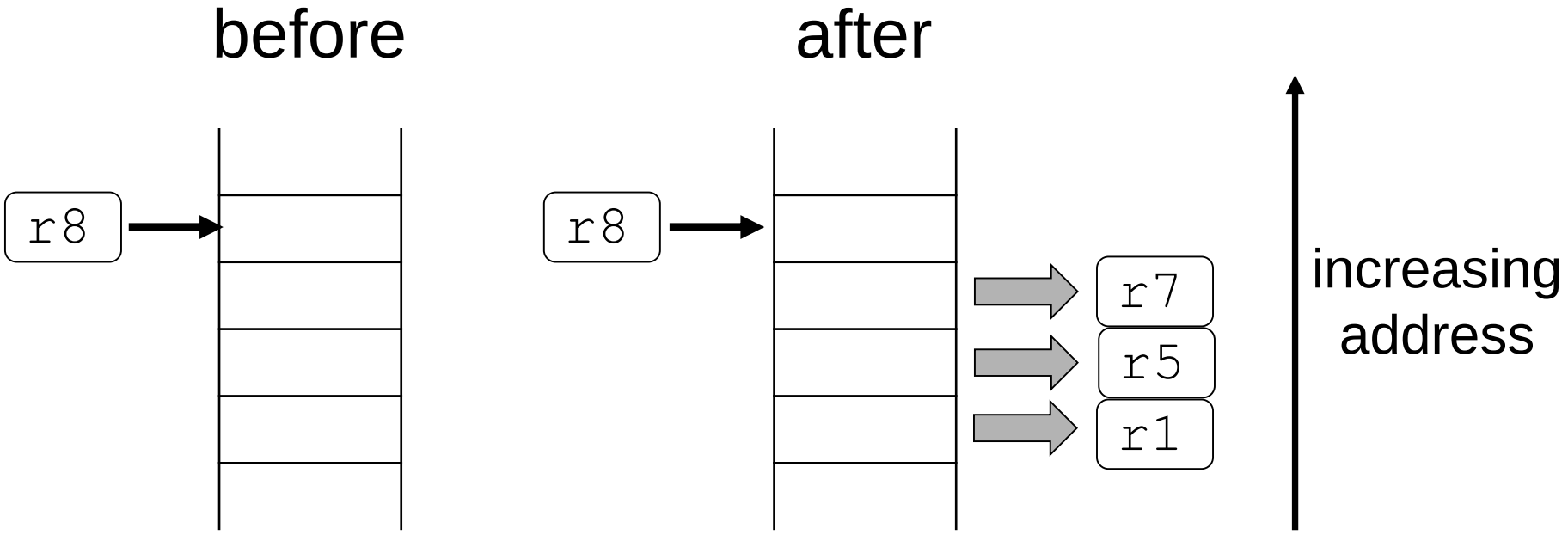
STMIA with ‘!’: an example

```
STMIA r8!, {r1, r5, r7}
STM r8!, {r1, r5, r7}
```



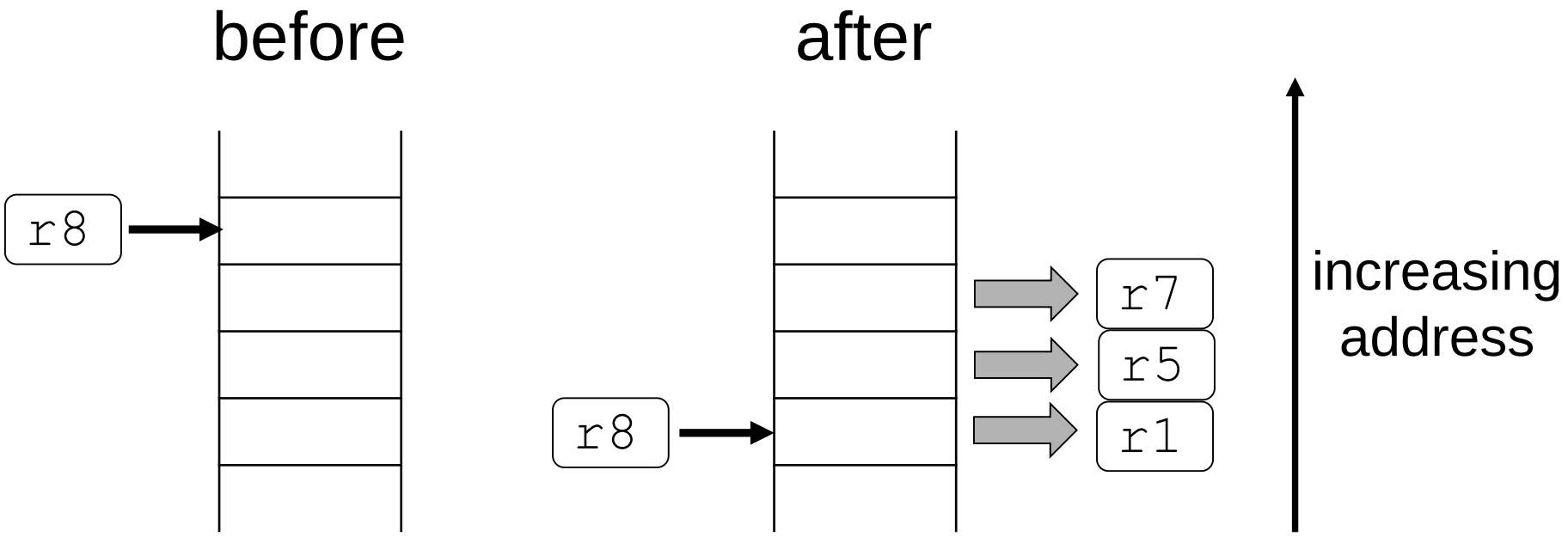
LDMDB: an example

```
LDMDB r8, {r1, r5, r7}
```



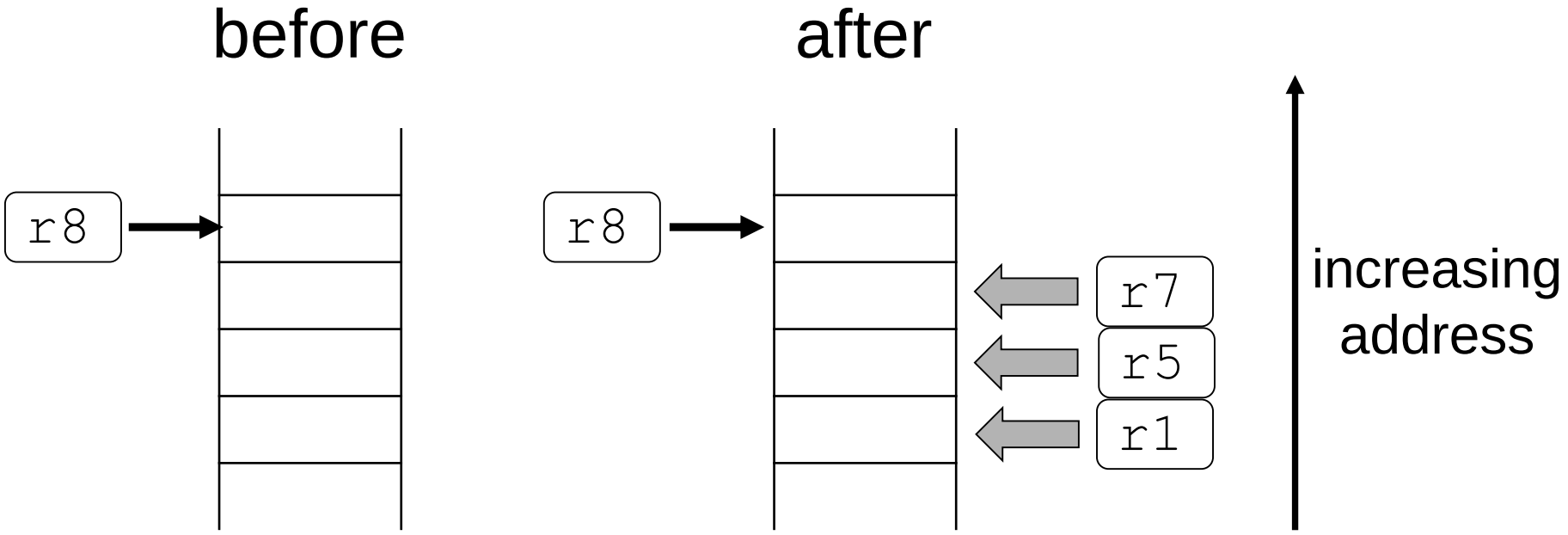
LDMDB with ‘!’: an example

```
LDMDB r8!, {r1, r5, r7}
```



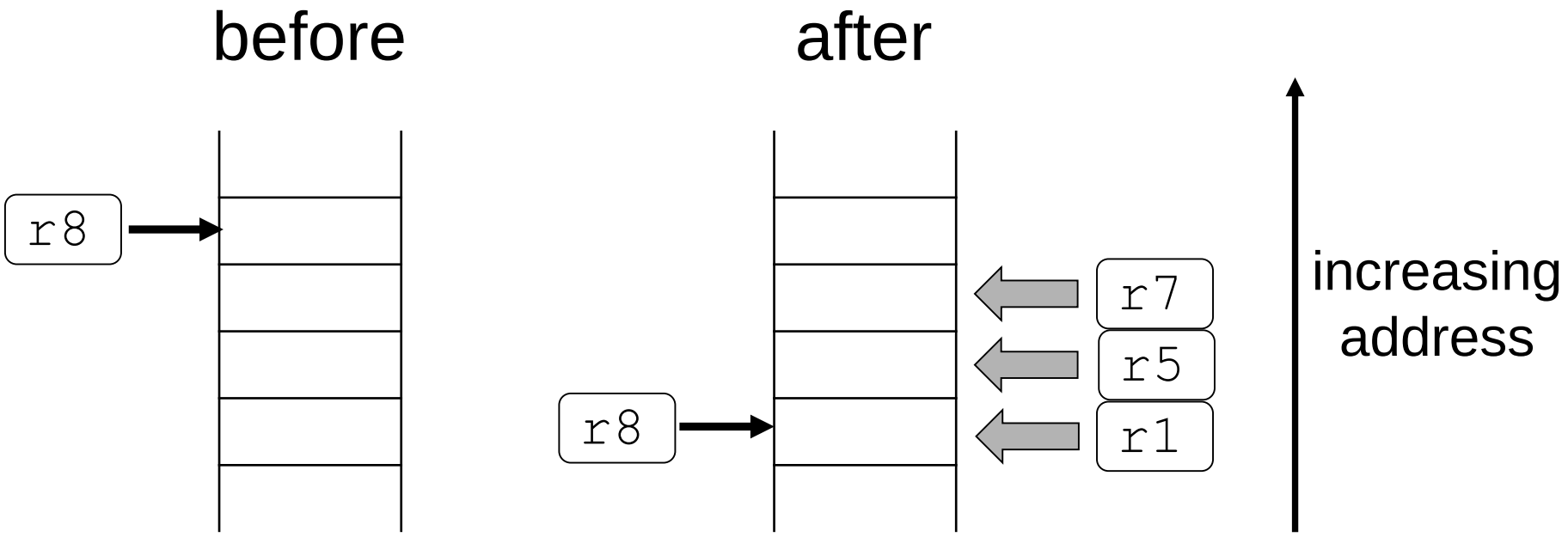
STMDB: an example

```
STMDB r8, {r1, r5, r7}
```



STMDB with ‘!’: an example

```
STMDB r8!, {r1, r5, r7}
```



Supported types of stack

- Stack-oriented suffixes can be used instead of increment/decrement and before/after.

Stack type	PUSH	POP
Full descending	STMDB STMFD	LDM LDMIA LDMFD
Empty ascending	STM STMIA STMEA	LDMDB LDMEA

PUSH and POP

- `PUSH` and `POP` instructions facilitate the use of a full descending stack.
- `PUSH <regList>` is the same as `STMDB SP!, <regList>`
- `POP <regList>` is the same as `LDMIA SP!, <regList>`