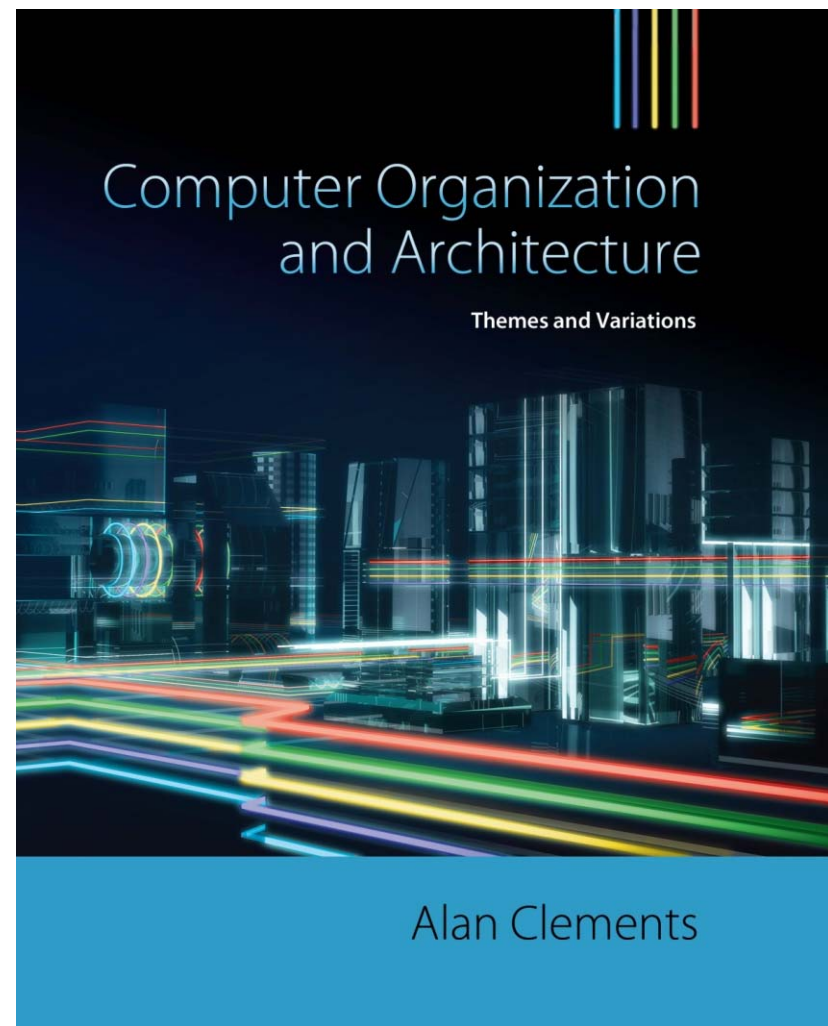


Part 0xB

CHAPTER 3

Architecture and Organization

1

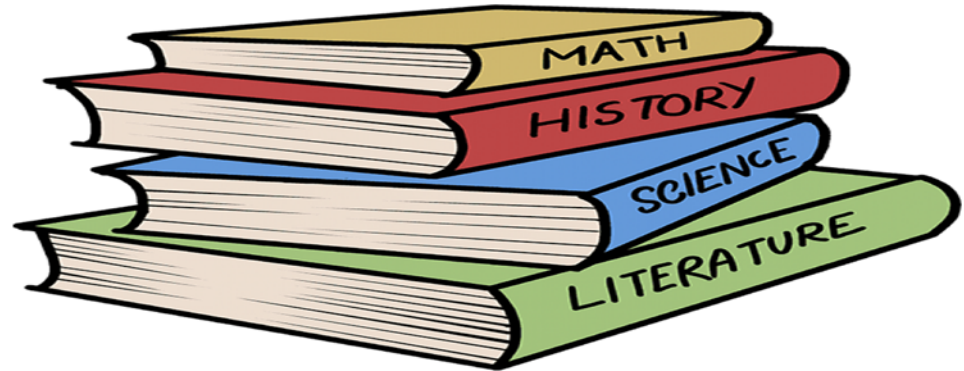


These slides are being provided with permission from the copyright for in-class (CS2208B) use only. The slides must not be reproduced or provided to anyone outside of the class.

All download copies of the slides and/or lecture recordings are for personal use only. Students must destroy these copies within 30 days after receipt of final course evaluations.

The Stack

- ❑ The stack is a data structure, a **last in first out** queue, **LIFO**, in which items **enter at one end** and **leave from the same end** in a **reverse order**.



- ❑ Stacks in microprocessors are implemented by using a **stack pointer** to point to the **top of the stack (TOS)** in memory.
- ❑ As items are
 - added (**pushed**) onto the stack, the stack pointer is moved **forward**, or
 - removed (**popped**) from the stack, the stack pointer is moved **backward**
- ❑ There are four ways of constructing a stack.
(See Figure 3.45 over the coming 4 slides)

Grows up

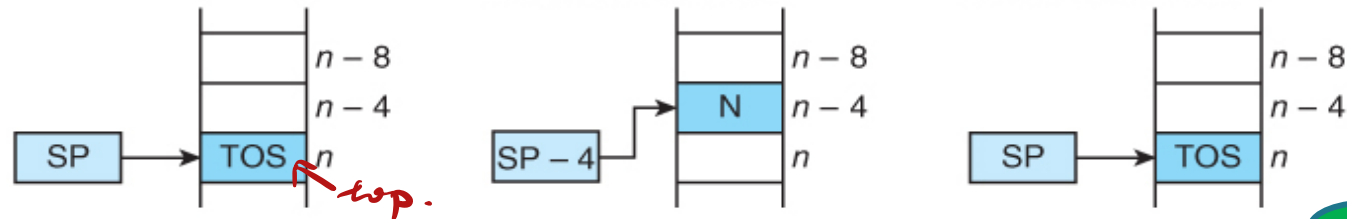
Occupied memory

The Stack

Initial state of the stack

Each stack's entry here is 4 bytes.

(a) Stack grows up.
Stack pointer points to TOS.



Pre-update

PUSH: SUB **SP**, #4 ; [SP] \leftarrow [SP] - 4 Adjust the stack pointer
STR R0, [**SP**] ; [[SP]] \leftarrow data Push data onto the stack

or simply

STR R0, [SP, #-4]! *store r0 in the address and before storing, moving SP 4 positions backwards.*

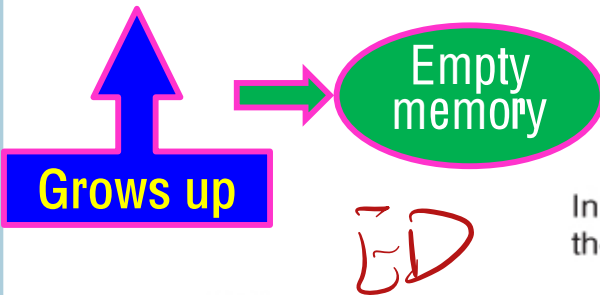
POP: LDR **R0**, [SP] ; data \leftarrow [[SP]] Pull data off the stack
ADD **SP**, #4 ; [SP] \leftarrow [SP] + 4 Adjust the stack pointer

or simply

LDR **R0**, [SP], #4

Post-update

TOS means *top of stack*

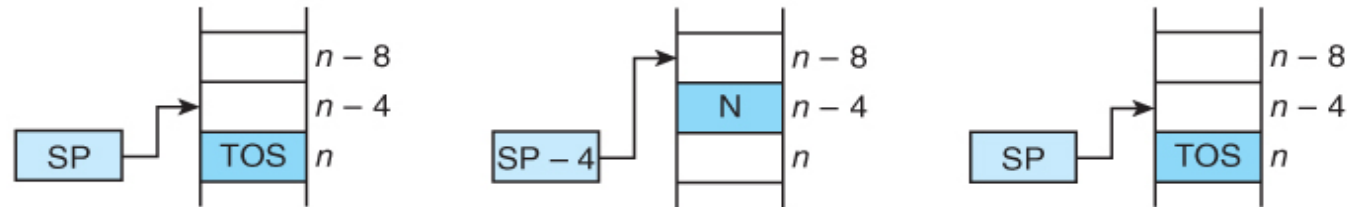


The Stack

Initial state of the stack

Each stack's entry here is 4 bytes.

(b) Stack grows up.
Stack pointer points to first free space.



PUSH: STR R0, [SP] ; [[SP]] ← data Push data onto the stack
 SUB SP, #4 ; [SP] ← [SP] - 4 Adjust the stack pointer

or simply

STR R0, [SP], #-4

Post-update

Pre-update

POP: ADD SP, #4 ; [SP] ← [SP] + 4 Adjust the stack pointer
 LDR R0, [SP] ; data ← [[SP]] Pull data off the stack

or simply

LDR R0, [SP, #4] !

TOS means *top of stack*

Grows down

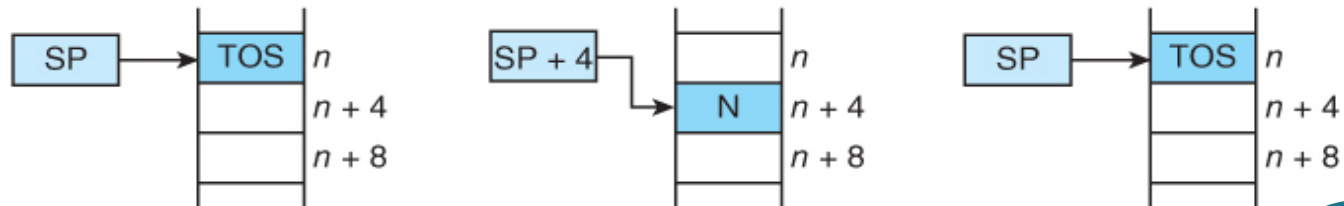
Occupied memory

The Stack

Initial state of the stack

Each stack's entry here is 4 bytes.

(c) Stack grows down. Stack pointer points to TOS.

*Pre-update*

PUSH: `ADD SP, #4` ; `[SP] ← [SP] + 4` Adjust the stack pointer
`STR R0, [SP]` ; `[[SP]] ← data` Push data onto the stack

or simply`STR R0, [SP, #4] !`

POP: `LDR R0, [SP]` ; `data ← [[SP]]` Pull data off the stack
`SUB SP, #4` ; `[SP] ← [SP] - 4` Adjust the stack pointer

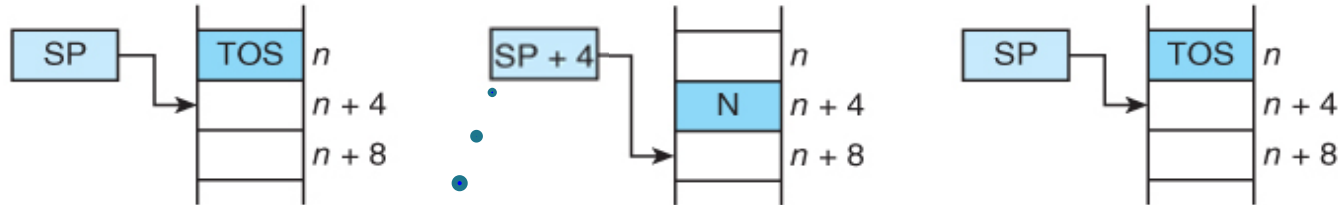
or simply`LDR R0, [SP], #-4`*Post-update***TOS** means *top of stack*

Grows down

EA

Empty
memory

The Stack

Initial state of
the stack*Each stack's entry here is 4 bytes.*(d) Stack grows down.
Stack pointer points
to first free space.*It is SP+4, not SP+8*

PUSH: STR R0, [SP] ; [[SP]] ← data Push data onto the stack
 ADD SP, #4 ; [SP] ← [SP] + 4 Adjust the stack pointer

or simply

STR R0, [SP], #4

*Post-update**Pre-update*

POP: SUB SP, #4 ; [SP] ← [SP] - 4 Adjust the stack pointer
 LDR R0, [SP] ; data ← [[SP]] Pull data off the stack

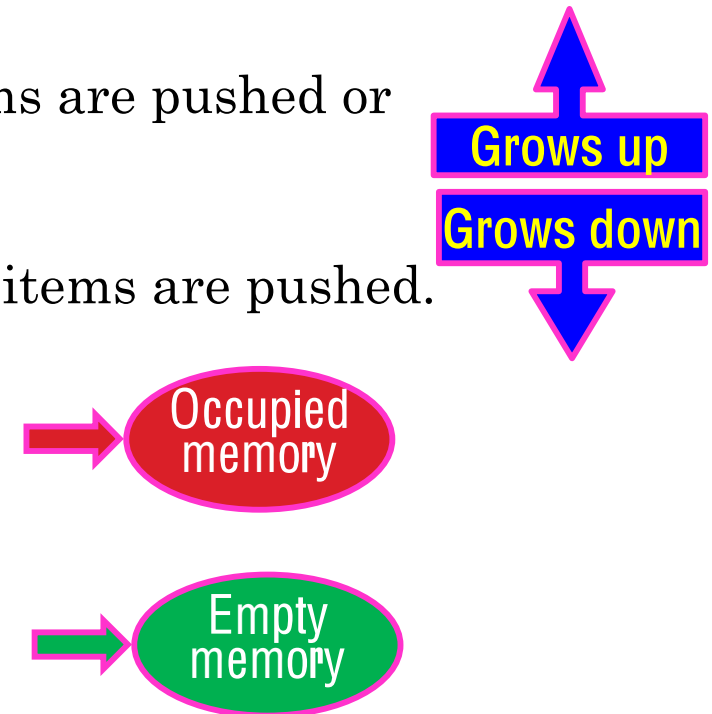
or simply

LDR R0, [SP, #-4] !

TOS means *top of stack*

The Stack

- ❑ The *two decisions* need to be made when implementing a stack are
- whether the stack grows
 - *up toward low memory addresses* as items are pushed or
 - *down toward high memory addresses* as items are pushed.
 - whether the stack pointer points to
 - the *top item* on the stack or
 - the *first free empty space* on the stake.



The Stack

- ❑ **CISC** processors automatically maintain the stack.
- ❑ **RISC** processors force the programmer to maintain the stack.