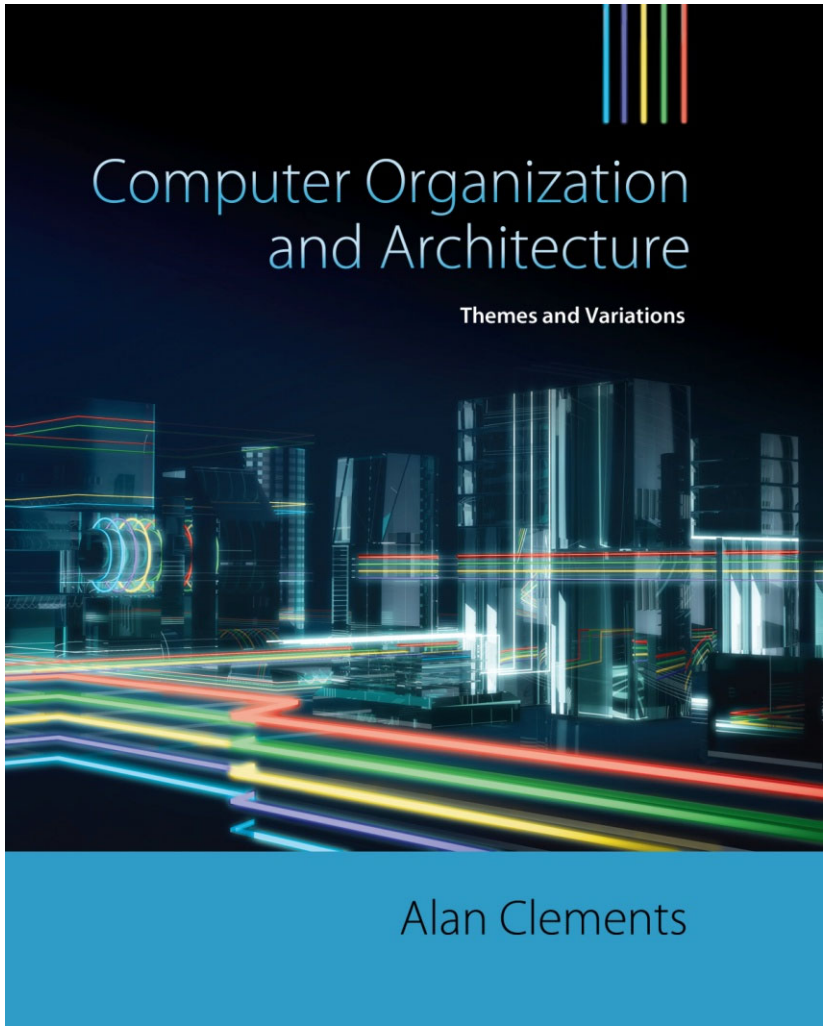
Part B

Chapter 3

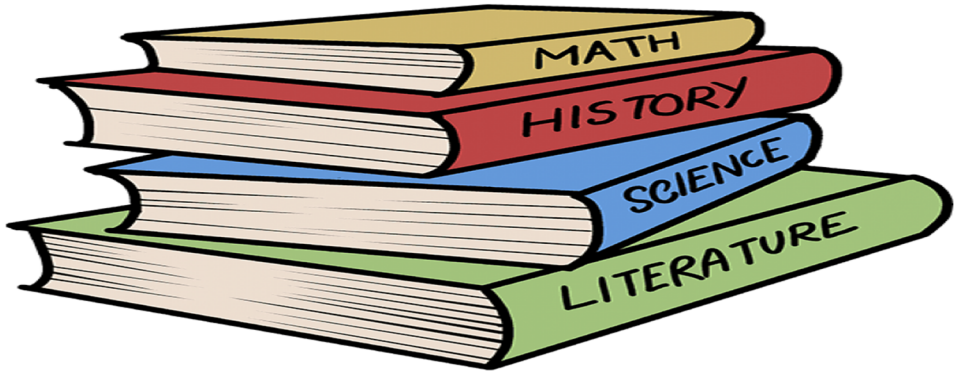
Architecture and Organization



CENGAGE Learning

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, depending on the definition of the ***top of the stack (TOS)*** and the definition of the ***forward*** / ***backward***. (*See Figure 3.45 over the coming 4 slides*)

Grows up

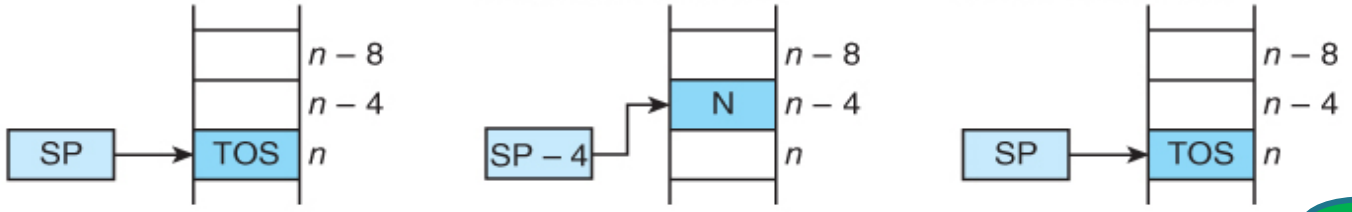
Occupied memory

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]  ← [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
       ADD SP,#4   ;[SP] ←  [SP] + 4   Adjust the stack pointer
```

or simply

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

Post-update

You need to re-map the memory to make the stack space read/write enabled (Debug/Memory Map). The other option is to use a .ini file
You may want to review tutorial 7, slides 93-106.

TOS means ***top of stack***

Grows up

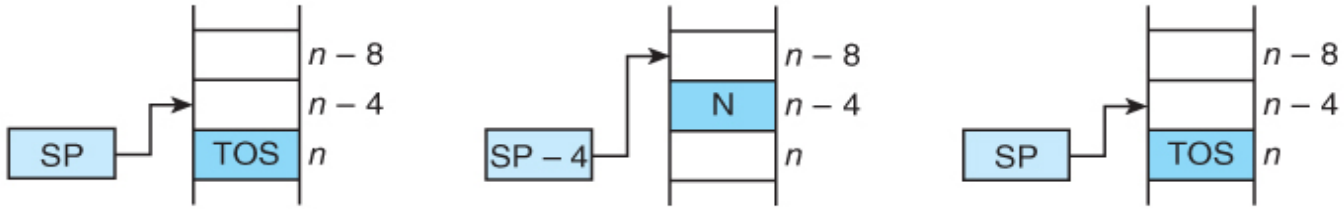
Empty memory

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***

You need to re-map the memory to make the stack space read/write enabled (Debug/Memory Map). The other option is to use a .ini file
You may want to review tutorial 7, slides 93-106.

Grows down

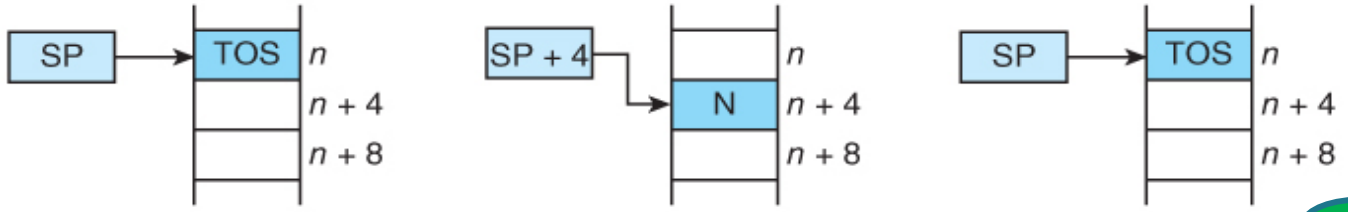
Occupied memory

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

You need to re-map the memory to make the stack space read/write enabled (Debug/Memory Map). The other option is to use a .ini file
You may want to review tutorial 7, slides 93-106.

TOS means ***top of stack***

Grows down

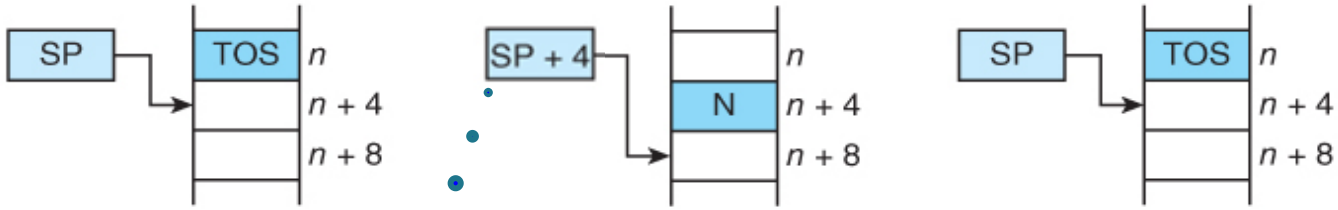
Empty memory

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*

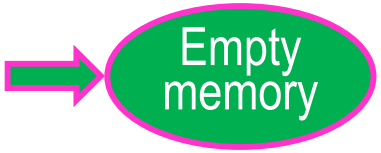
You need to re-map the memory to make the stack space read/write enabled (Debug/Memory Map). The other option is to use a .ini file
You may want to review tutorial 7, slides 93-106.

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, i.e., the word "***forward***" in slide 162 means ***decrement***, and the word "***backward***" in slide 162 means ***increment*** or
    - *down toward high memory addresses* as items are pushed, i.e., the word "***forward***" in slide 162 means ***increment***, and the word "***backward***" in slide 162 means ***decrement***.

  - whether the stack pointer points to
    - the *top item* on the stack or

    - the *first free empty space* on the stack.



The Stack

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