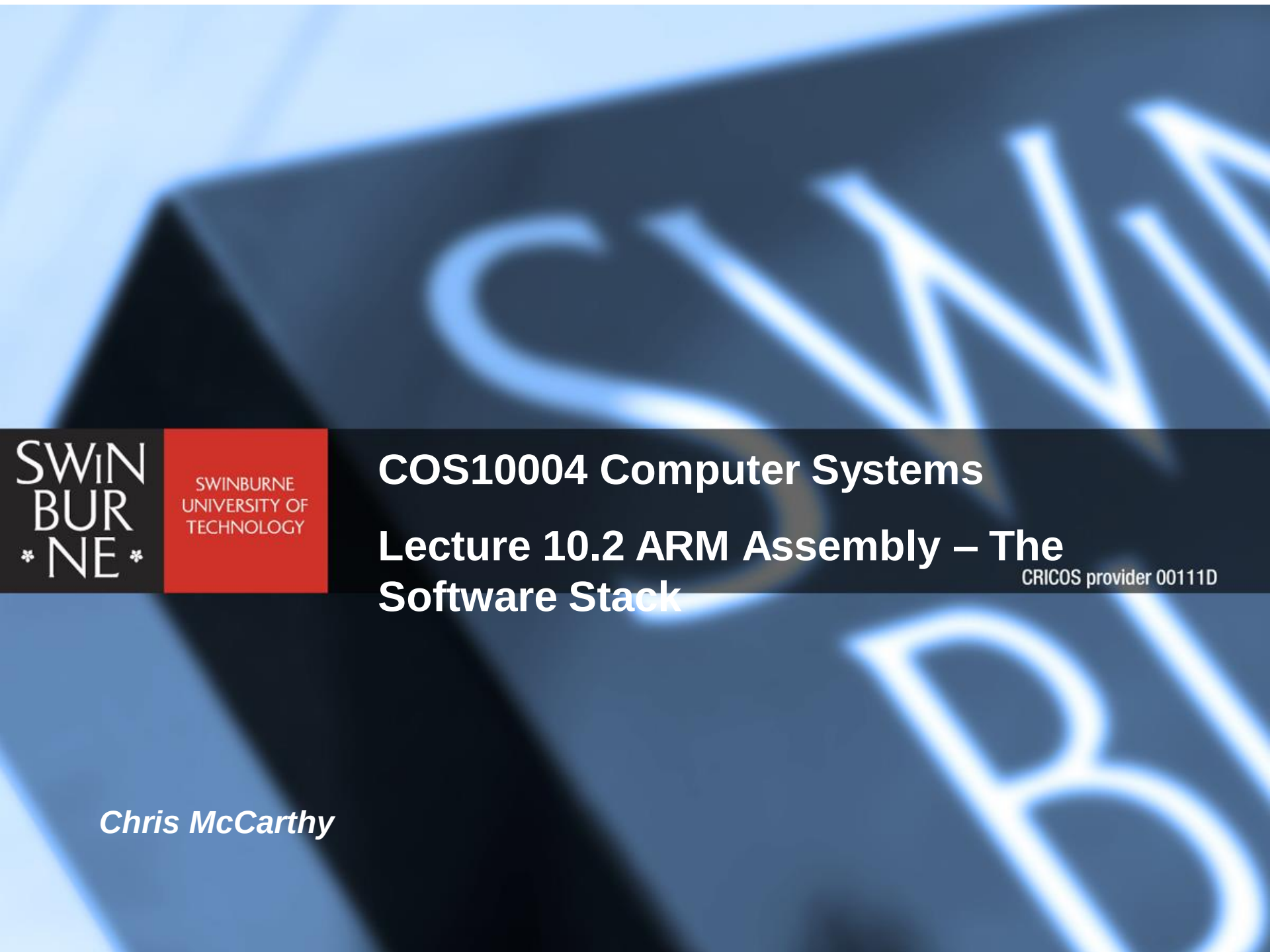


COS10004 Computer Systems

Lecture 10.2 ARM Assembly – The Software Stack

Chris McCarthy

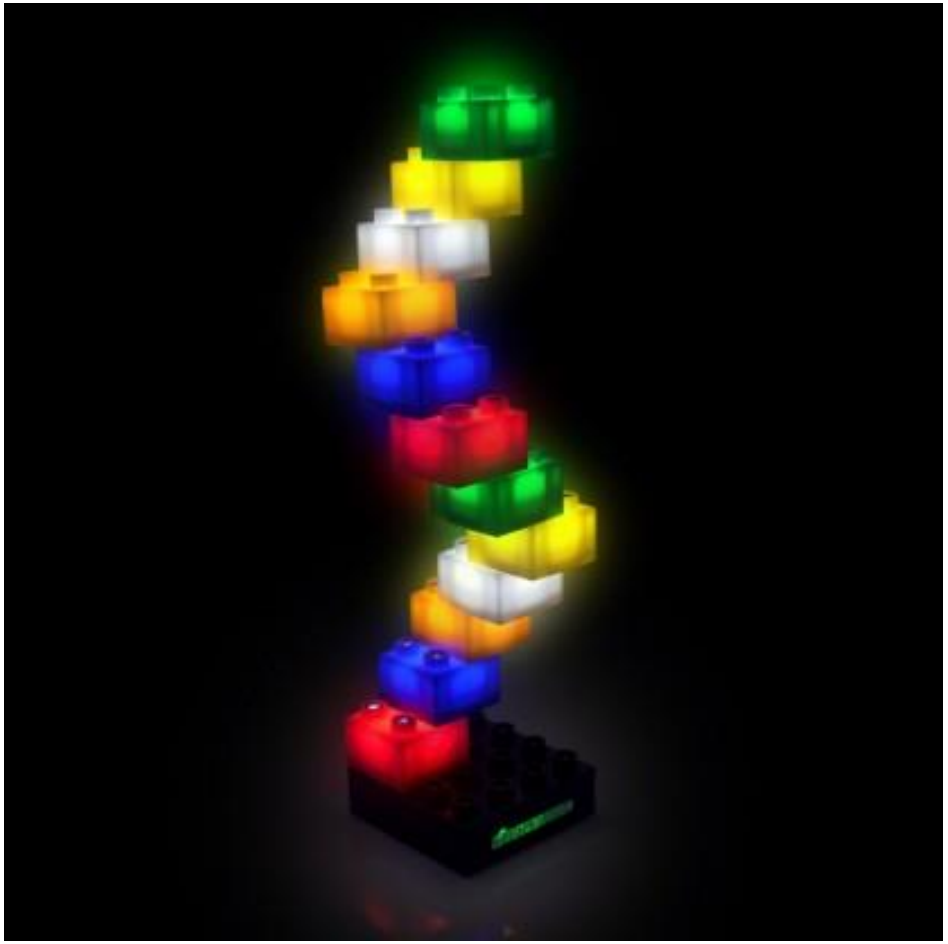
FUNCTIONS IN ASM

- Not 'native' to assembly
  - We need to do a lot of the management ourselves
- Argument passing:
  - How do we pass arguments from one function to another
- Storing and recalling register values
  - each function we call will want to use the same registers (only 13 general purpose registers !)
  - How do we manage this ?
- Managing the program control
  - Jumping from one function to another, and then returning back !

FUNCTIONS IN ASM

- Not 'native' to assembly
  - We need to do a lot of the management ourselves
- Argument passing:
  - How do we pass arguments from one function to another
- Storing and recalling register values
  - each function we call will want to use the same registers (only 13 general purpose registers !)
  - How do we manage this ?
- Managing the program control
  - Jumping from one function to another, and then returning back !

STACKS



www.glow.co.uk

PUSH, POP AND THE STACK

- ARM computers have a software stack*.
- A separate area of RAM is available for temporary values.
- A value in a register can be pushed onto the stack to preserve it for later.
- It can be popped off later (in LIFO order).
- We can get the memory location (a pointer to it) by *checking the SP* (R13) register.

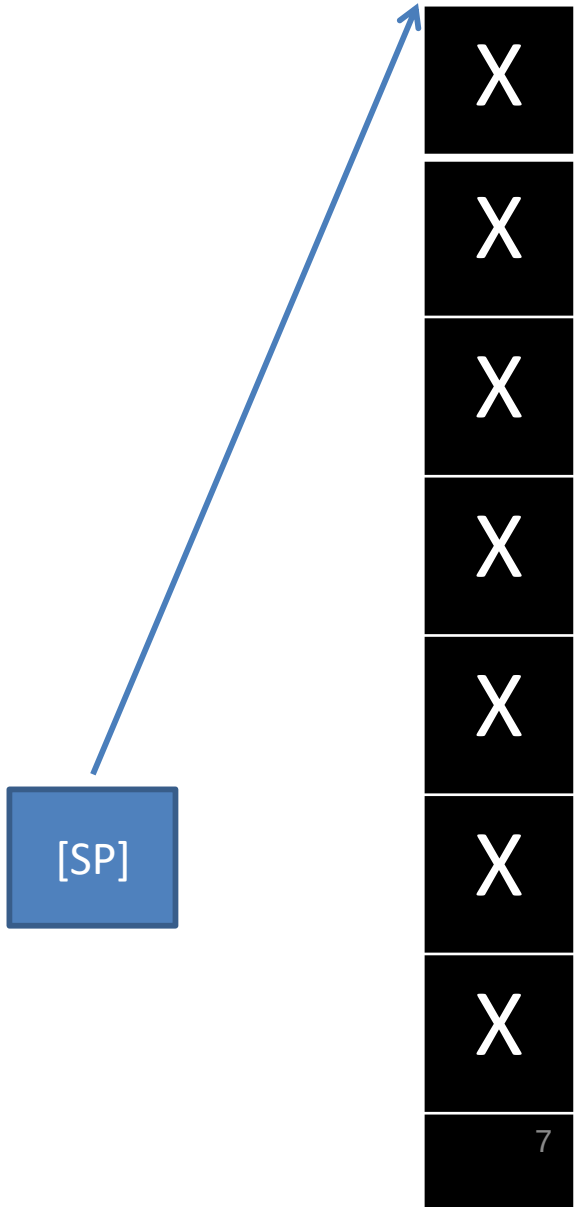
* SOFTWARE STACK?

- A section of RAM managed by the SP (stack pointer) register.
- A sort of 32-bit (64-bit in ARM8) wide array which starts (element 0) high in RAM and grows down as values are added to it.
- The stack pointer stores the memory location of the last value added (pushed) to the stack.
- Each push decrements SP by 4 (4 bytes per word).
- A pop operation removes the last value in the stack and increments the SP by 4 (4 bytes per word)

Software stack (depth only limited by RAM)

Example:

x = don’t care. [SP] points to start of stack.

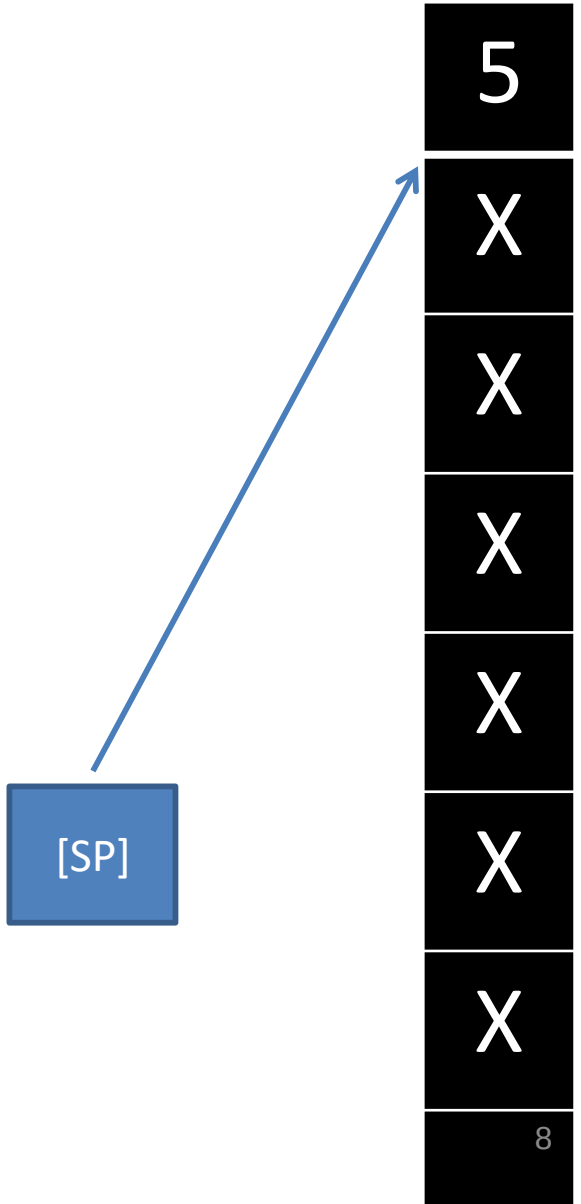


Software stack (depth only limited by RAM)

Example:

x = don't care.

push 5 – SP decremented by 4. [SP] points to 5



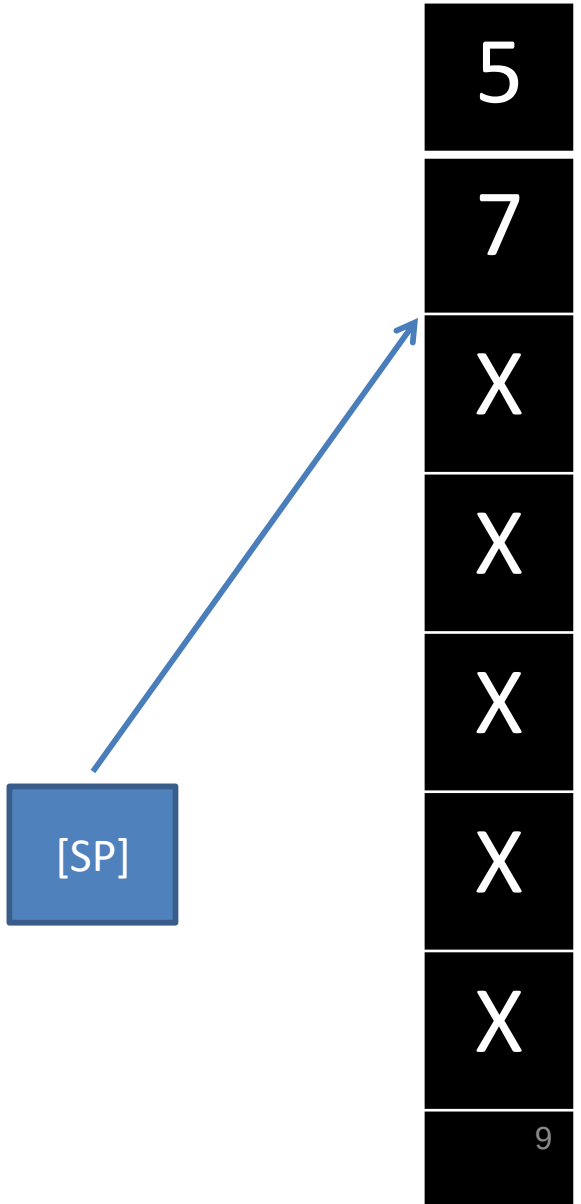
Software stack (32-bit) (depth only limited by RAM)

Example:

x = don't care.

push 5 – SP decremented by 4

push 7 - SP decremented by 4, [SP] points to 7.



Software stack (depth only limited by RAM)

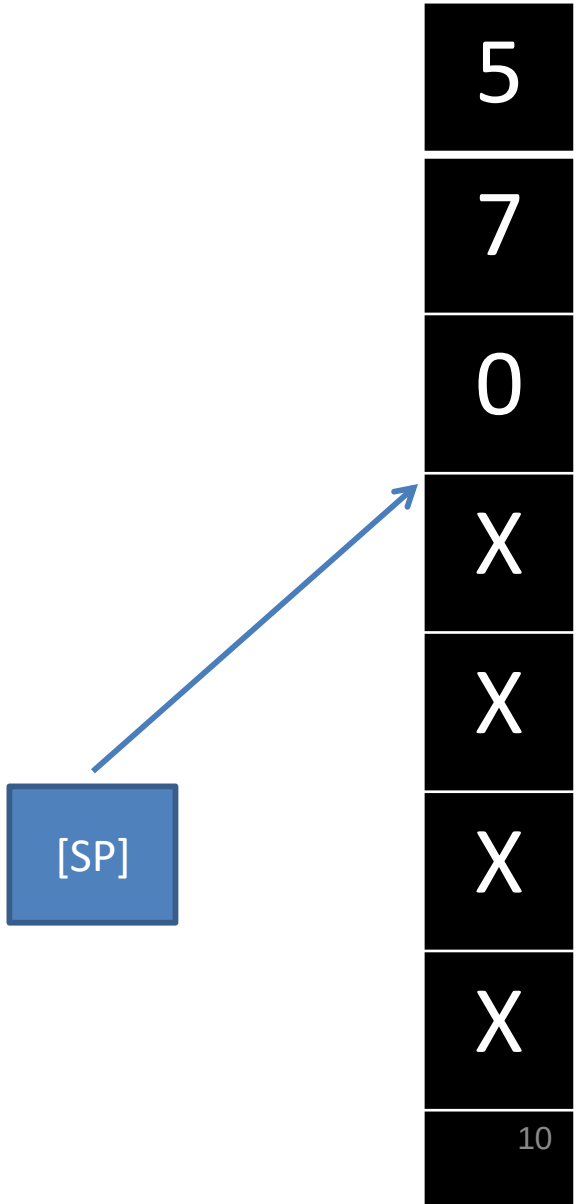
Example:

x = don't care.

push 5 – SP decremented by 4

push 7 - SP decremented by 4

push 0 – SP decremented by 4. [SP] points to 0



Software stack (depth only limited by RAM)

Example:

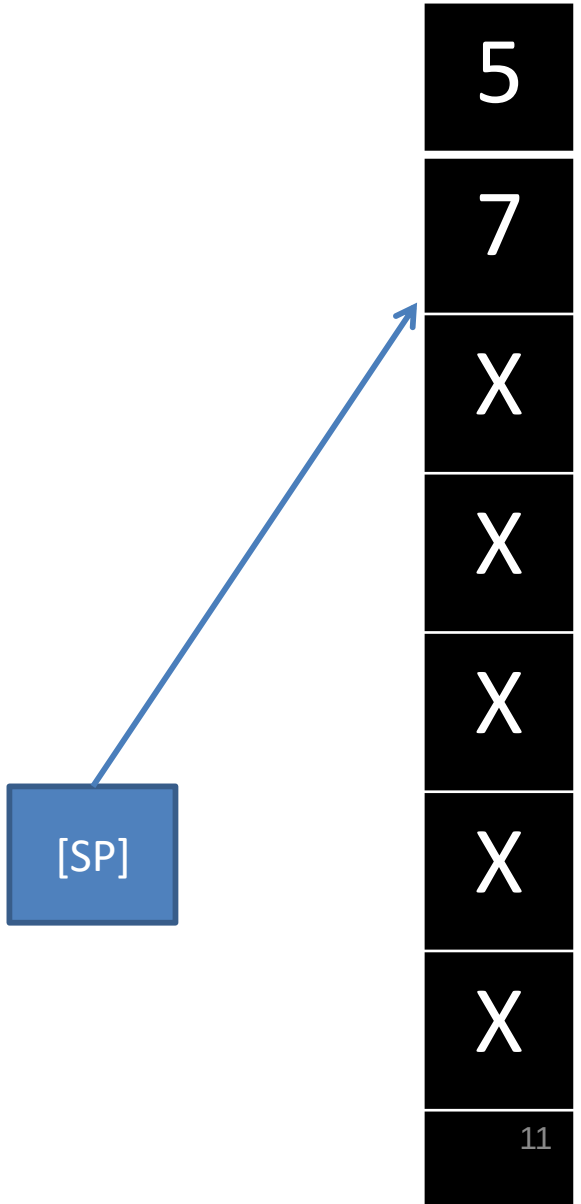
x = don't care.

push 5 – SP decremented by 4

push 7 - SP decremented by 4

push 0 – SP decremented by 4

pop 0 – SP incremented by 4. [SP] points to 7



Software stack (depth only limited by RAM)

Example:

x = don't care.

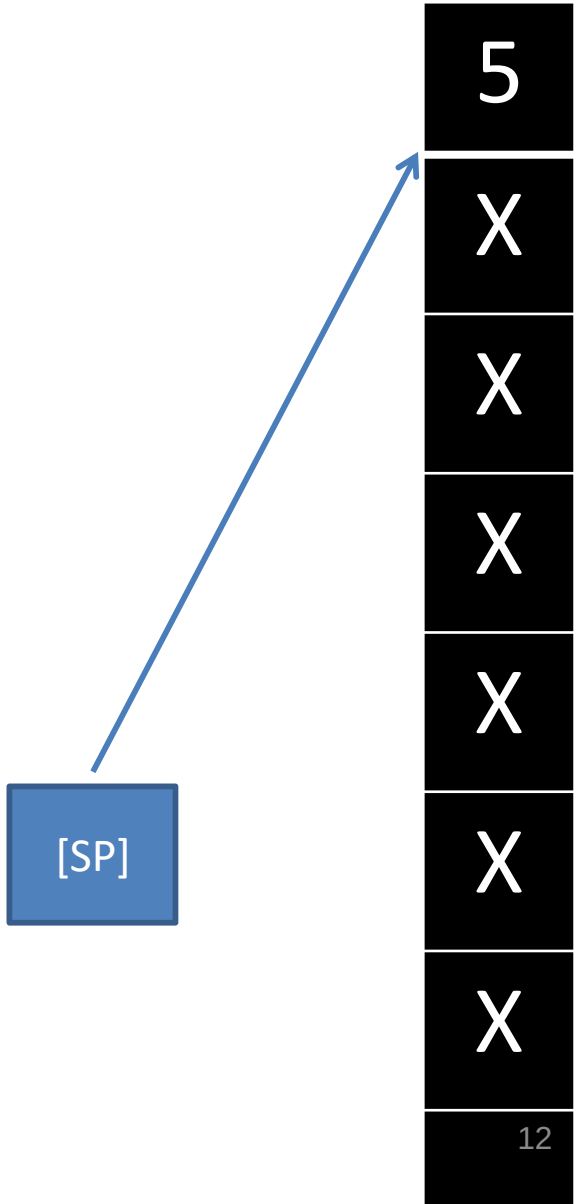
push 5 – SP decremented by 4

push 7 - SP decremented by 4

push 0 – SP decremented by 4

pop 0 – SP incremented by 4

pop 7 – SP incremented by 4. [SP] points to 5.



Software stack (depth only limited by RAM)

Example:

x = don't care.

push 5 – SP decremented by 4

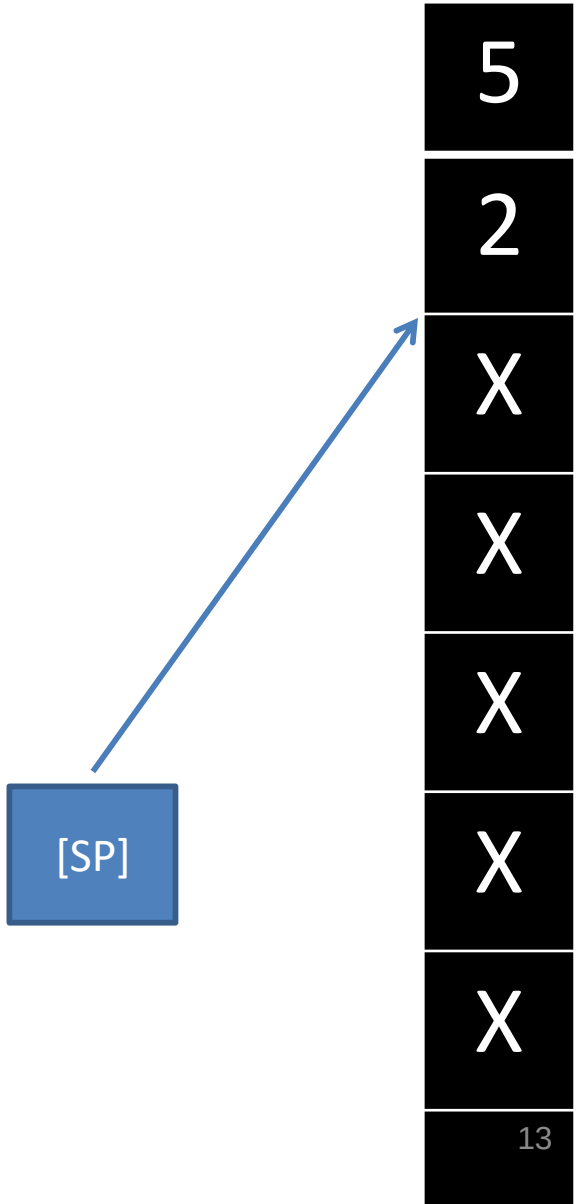
push 7 - SP decremented by 4

push 0 – SP decremented by 4

pop 0 – SP incremented by 4

pop 7 – SP incremented by 4

push 2 – SP decremented by 4. [SP] points to 2.



Software stack (depth only limited by RAM)

Example:

x = don't care.

push 5 – SP decremented by 4

push 7 - SP decremented by 4

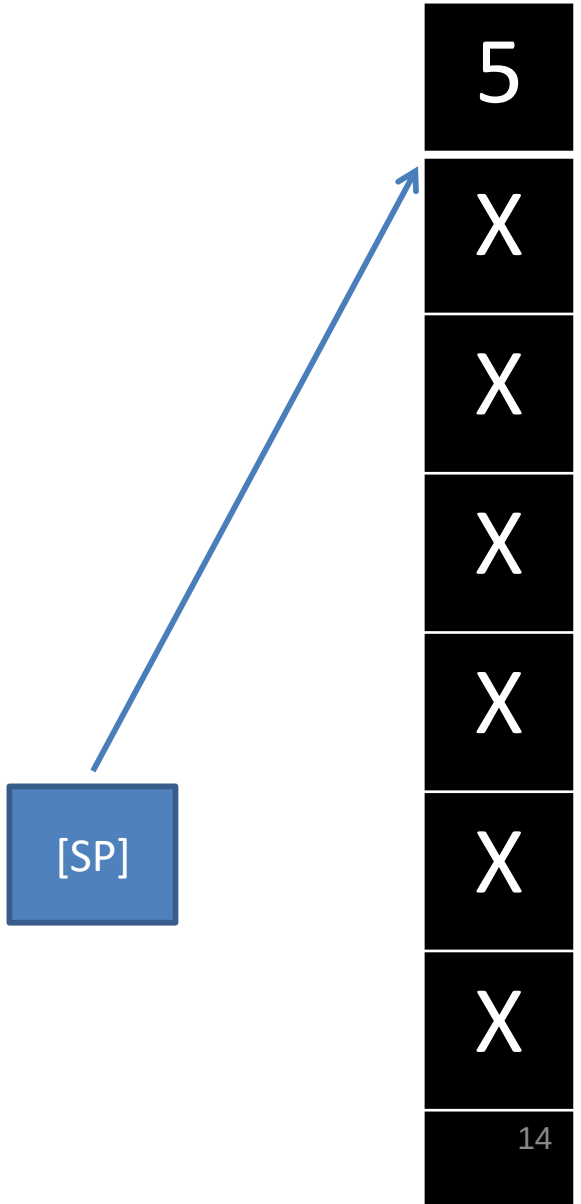
push 0 – SP decremented by 4

pop 0 – SP incremented by 4

pop 7 – SP incremented by 4

push 2 – SP decremented by 4

pop 2 – SP incremented by 4. [SP] points to 5.



Software stack (depth only limited by RAM)

Example:

x = don't care.

push 5 – SP decremented by 4

push 7 - SP decremented by 4

push 0 – SP decremented by 4

pop 0 – SP incremented by 4

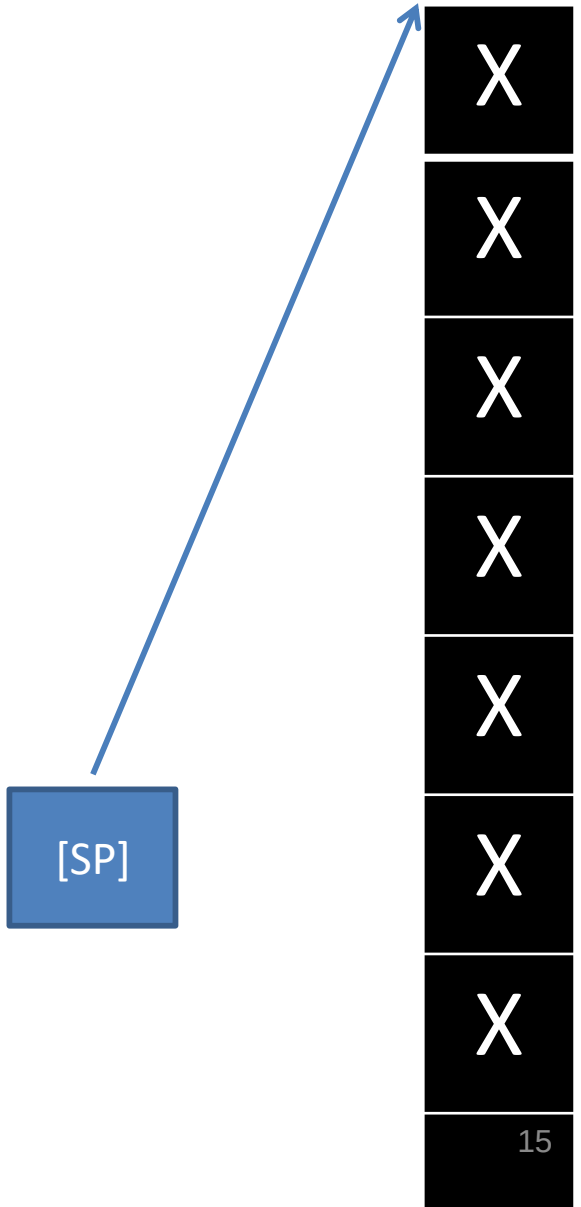
pop 7 – SP incremented by 4

push 2 – SP decremented by 4

pop 2 – SP incremented by 4

pop 5 - SP incremented back to it's starting value. The stack is now empty.

[SP] points to end of stack.



EXAMPLE SYNTAX

- Push and pop accept multiple registers if in a { , , ,...} list

```
push {r4,r5}  ;back them up onto the stack
;use r4 and r5  for something else
pop {r4,r5} ;restore them from the stack
```

Correct order is preserved for {lists}

- Alternatively, do one at a time (but pop in reverse order)

```
push {r4}
push {r5}
; do something
pop {r5}
pop {r4}
```

RECALL THE ABI

- **Application Binary Interface** (ABI) sets standard way of using ARM registers.
  - r0-r3 used for function arguments and return values
  - r4-r12 promised not to be altered by functions
  - **lr** and **sp** used for stack management
  - **pc** is the next instruction – we can use it to exit a function call

ABI Conventions

- ABI compliant functions:
  - Use r0-r3 for passing and returning values to functions
  - Promise not to alter r4-r12
- … but suppose the function needs to use many registers to do calculations ??
- We can use the stack to store and recall register values !

Passing Arguments to functions

- To re-use the registers we need to:
  - Back up registers we need to re-use in a function
  - Store arguments for the function in r0-r3
  - Call the function
  - Read the return values from r0-r1 (optional)
  - Restore the registers we backed up.

SUMMARY

- Software Stack:
  - Dedicated RAM used to store values FILO
    - Special register “sp” used to store address of start of the stack
- Stacks allow us to store and recall register values efficiently
- Stacks integral to functions:
  - We need to store and recall register values so we don’t run out of registers to use!