SNS COLLEGE OF ENGINEERING

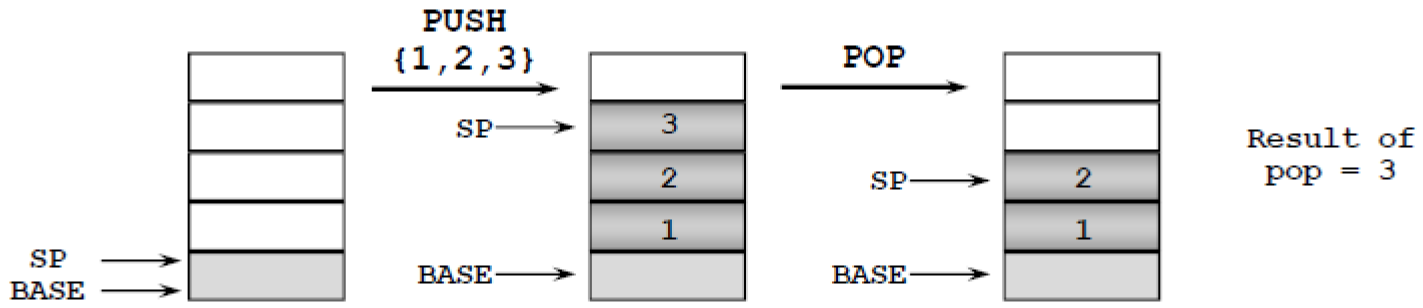
(Autonomous)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

STACK AND SUB- ROUTINES

Stacks

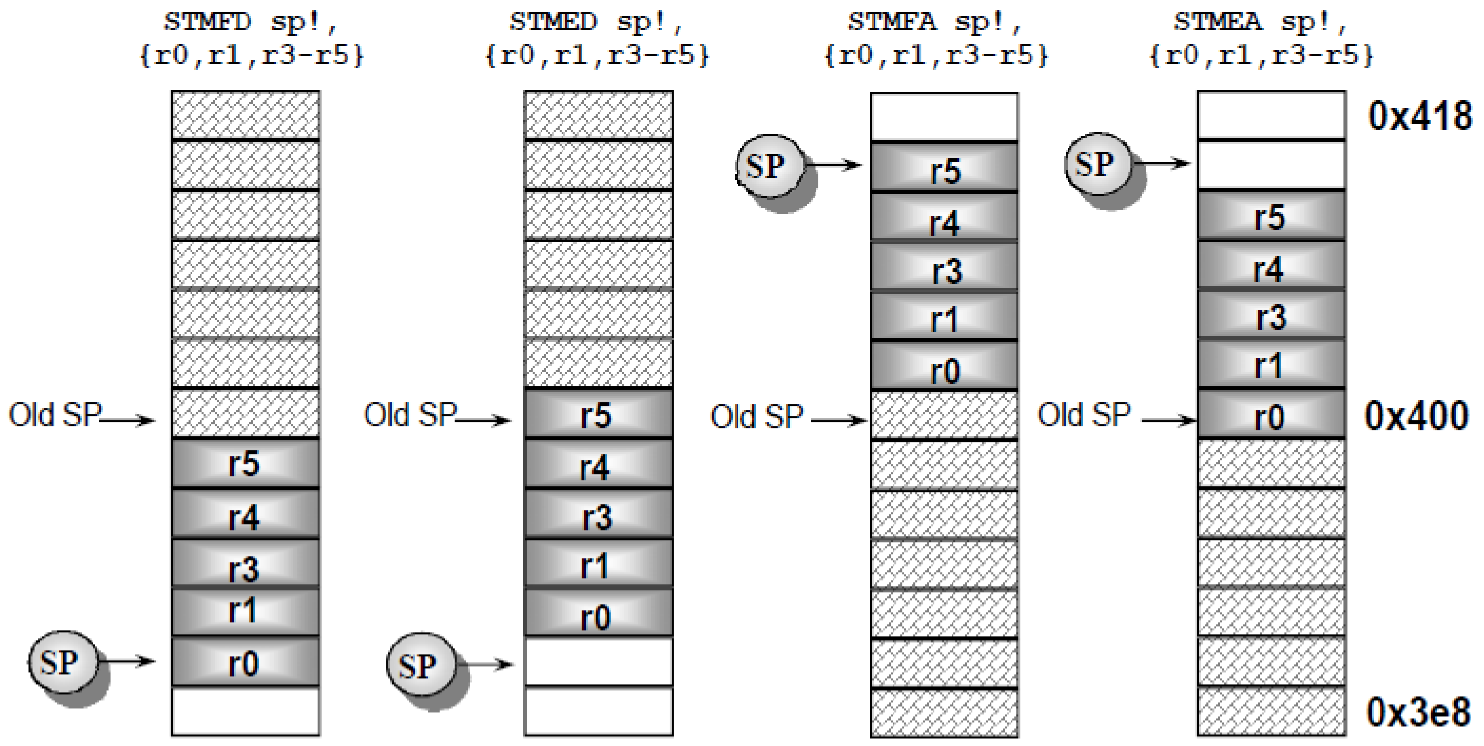
- A stack is an area of memory which grows as new data is "pushed" onto the "top" of it, and shrinks as data is "popped" off the top.
- Two pointers define the current limits of the stack.
  - A base pointer
    - used to point to the "bottom" of the stack (the first location).
  - A stack pointer
    - used to point the current "top" of the stack.



Stack Operation

- Traditionally, a stack grows down in memory, with the last "pushed" value at the lowest address. The ARM also supports ascending stacks, where the stack structure grows up through memory.
- The value of the stack pointer can either:
  - Point to the last occupied address (Full stack)
    - and so needs pre-decrementing (ie before the push)
  - Point to the next occupied address (Empty stack)
    - and so needs post-decrementing (ie after the push)
- The stack type to be used is given by the postfix to the instruction:
  - STMFD / LDMFD : Full Descending stack
  - STMFA / LDMFA : Full Ascending stack.
  - STMED / LDMED : Empty Descending stack
  - STMEA / LDMEA : Empty Ascending stack
- Note: ARM Compiler will always use a Full descending stack.

Stack Examples



Stacks and Subroutines

- One use of stacks is to create temporary register workspace for subroutines. Any registers that are needed can be pushed onto the stack at the start of the subroutine and popped off again at the end so as to restore them before return to the caller :

```
STMFD sp!,{r0-r12, lr}        ; stack all registers
........                      ; and the return address
........
LDMFD sp!,{r0-r12, pc}        ; load all the registers
                              ; and return automatically
```

- See the chapter on the ARM Procedure Call Standard in the SDT Reference Manual for further details of register usage within subroutines.
- If the pop instruction also had the 'S' bit set (using '^') then the transfer of the PC when in a privileged mode would also cause the SPSR to be copied into the CPSR (see exception handling module).

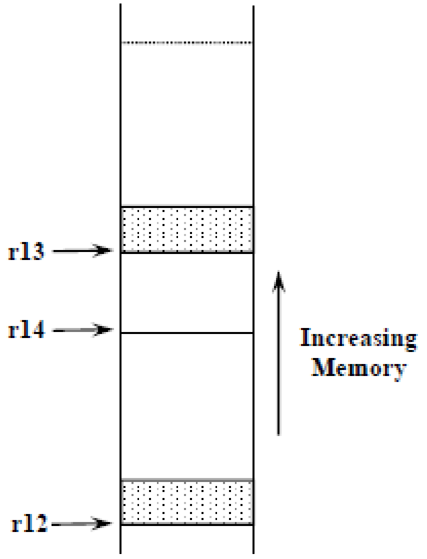
Direct functionality of Block Data Transfer

- When LDM / STM are not being used to implement stacks, it is clearer to specify exactly what functionality of the instruction is:
  - i.e. specify whether to increment / decrement the base pointer, before or after the memory access.
- In order to do this, LDM / STM support a further syntax in addition to the stack one:
  - STMIA / LDMIA : Increment After
  - STMIB / LDMIB : Increment Before
  - STMDA / LDMDA : Decrement After
  - STMDB / LDMDB : Decrement Before

Example: Block Copy

- Copy a block of memory, which is an exact multiple of 12 words long from the location pointed to by r12 to the location pointed to by r13. r14 points to the end of block to be copied.

```
; r12 points to the start of the source data
; r14 points to the end of the source data
; r13 points to the start of the destination data
loop   LDMIA  r12!, {r0-r11} ; load 48 bytes
       STMIA  r13!, {r0-r11} ; and store them
       CMP    r12, r14       ; check for the end
       BNE    loop           ; and loop until done
```



- This loop transfers 48 bytes in 31 cycles
- Over 50 Mbytes/sec at 33 MHz