COMSM1302
Overview of Computer Architecture

Lecture 14
ARM Memory Access Instructions

In the previous lecture

- Multiplication and division instructions.
- Shifting operations.
- Fixed point math.
- Example: Newton and Raphson’s method.

In this lecture

LDR/STR
- Load constant and base register
- Addressing modes
- Array access

LDM/STM
- Block data transfer
- Stack

- At the end of this lecture:
  - Learn how to load big constant numbers.
  - How to load and store data to and from memory.
  - How stacks are implemented in ARM.

Load / Store instructions

- The ARM has three sets of instructions which interact with main memory.
  - Single register data transfer (LDR / STR).
  - Block data transfer (LDM/STM).
  - Single Data Swap (SWP).

ARM memory access instructions

LDR/STR

- Load constant and base register
- Addressing modes
- Array access

LDM/STM

- Block data transfer
- Stack

Single register data transfer

- Syntax:
  - <LDR|STR>{<cond>} Rd, <address>
- Operation
  - ldr loads the content of the memory location at the given address to the register Rd.
  - srt saves the content of register Rd to the memory location at the given address.

LDR/STR

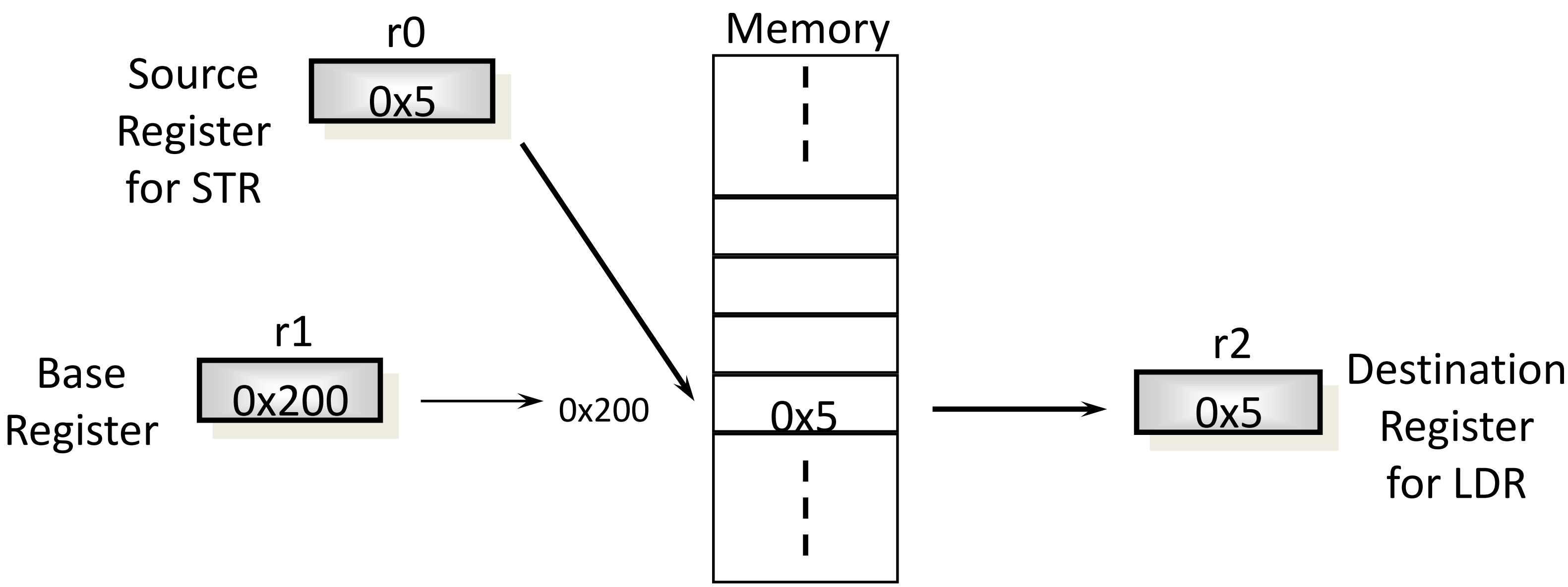
- Load constant and base register
- Addressing modes
- Array access

Loading full 32 bit constants

- `LDR rd,=numeric constant`
  - `LDR r0,=0x42`
    - `Assembler generates MOV r0,#0x42`
  - `LDR r0,=0x55555555`
    - `Assembler generate LDR r0,[pc, offset to a literal pool]`
- This mechanism will always generate the best instruction for a given case, thus, it is the recommended way of loading constants.

Load and Store : Base register

- The memory location to be accessed is held in a base register
  - STR r0, [r1]   LDR r2, [r1]

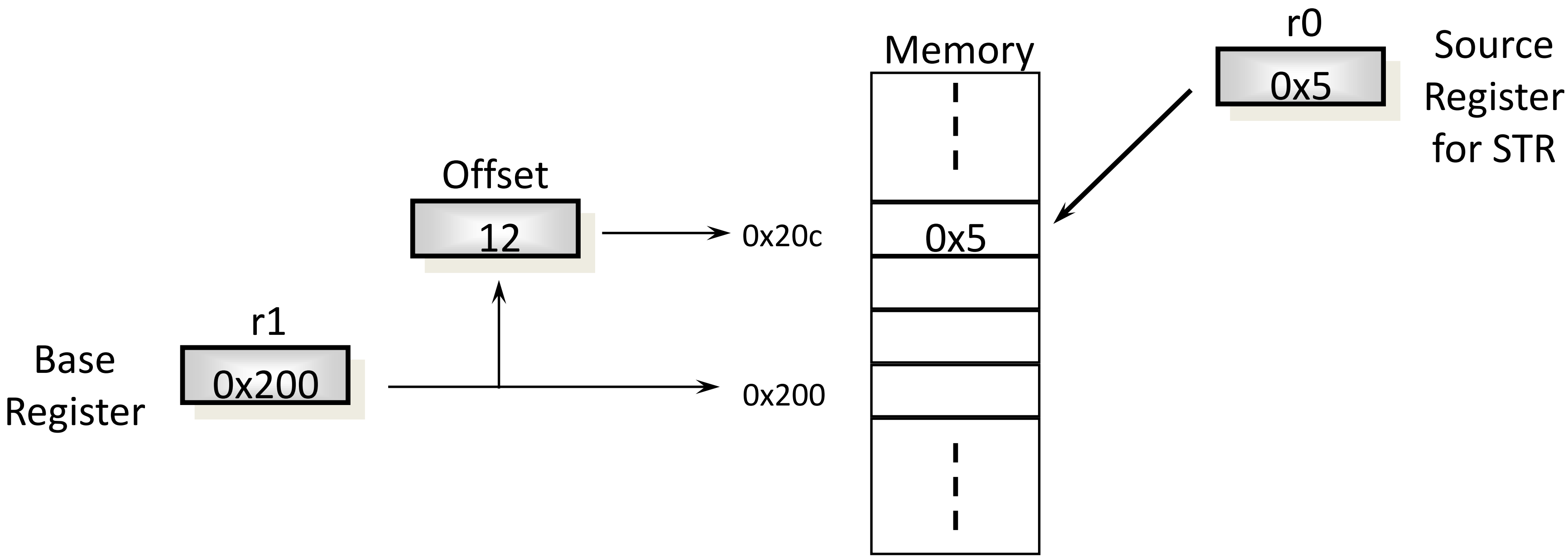


LDR/STR

- Load constant and base register
- Addressing modes
- Array access

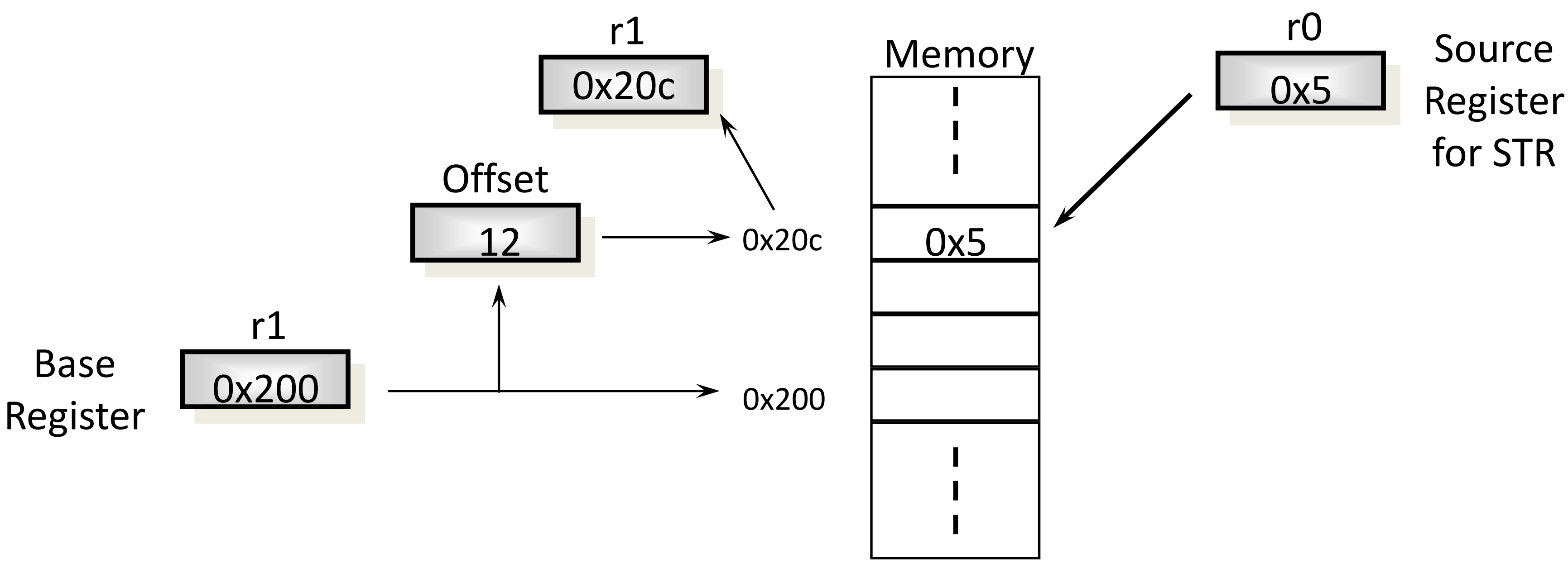
Load and Store – pre-indexed addressing

- Access a location offset from the base register
- Example: STR r0, [r1,#12]



Load and Store - pre-indexed addressing with right back

- Access a location offset from the base register
- Example: STR r0, [r1,#12]!



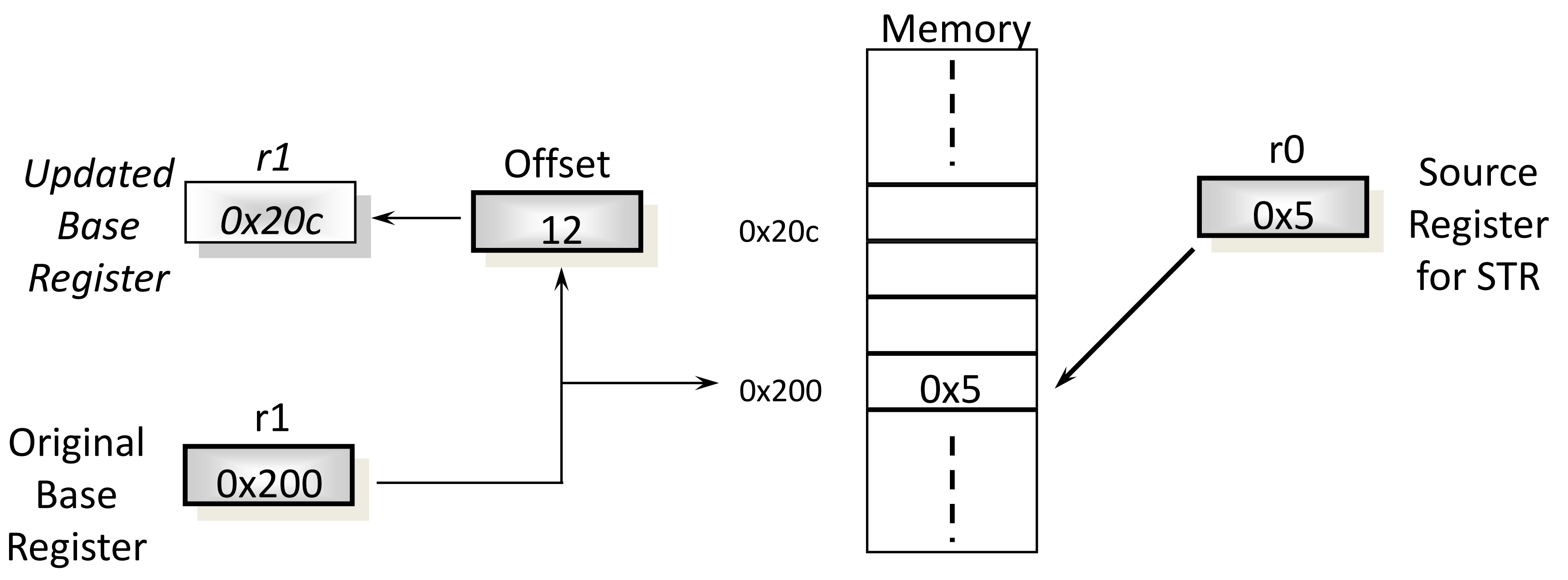
Load and Store - pre-indexed addressing - questions

- If r1 = 0x200, what is the address where r0 will be stored, and what is the value of r1 after executing the following instructions?

1. STR r0, [r1,#-12]
2. STR r0, [r1, #-12]!
3. If r2 contains 12, STR r0, [r1, r2]
4. If r2 contains 3, STR r0, [r1, r2, LSL #2]

Load and Store - post-indexed addressing

- Example: STR r0, [r1], #12

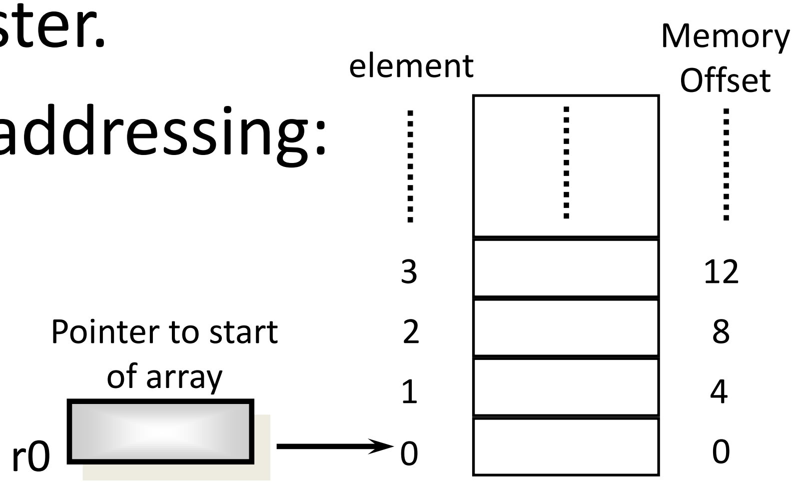


Load and Store - post-indexed addressing - questions

- If r1 = 0x200, what is the address where r0 will be stored, and what is the value of r1 after executing the following instructions?
  - STR r0, [r1], #-12
  - If r2 contains 12, STR r0, [r1], r2
  - If r2 contains 3, STR r0, [r1], r2, LSL #2
- Why we do not have “STR r0, [r1], #12!” ?

Usage of addressing modes

- If r0 points to the first element of an array in the memory.
- How can we access a particular element?
- Let r1 contains the index of the required element, and r2 be the destination register.
- Then we can use pre-indexed addressing:
  - LDR r2, [r0, r1, LSL #2]

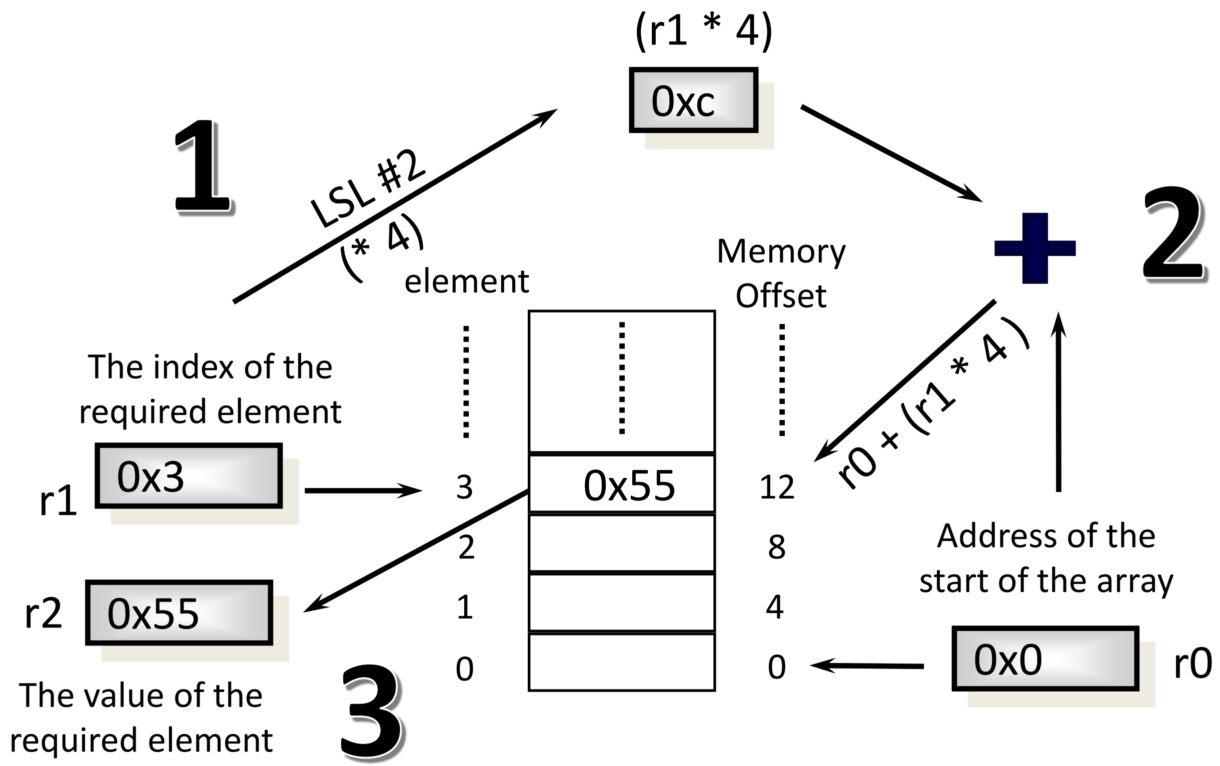


LDR/STR

- Load constant and base register
- Addressing modes
- Array access

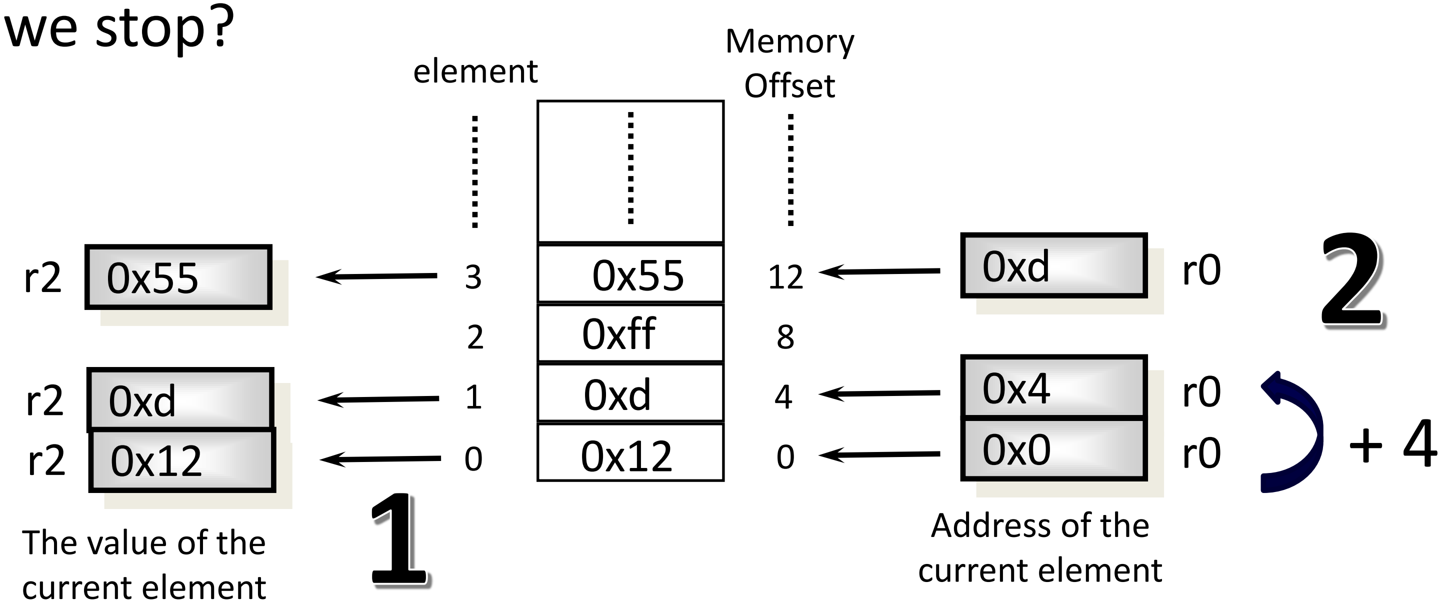
Usage of addressing modes – direct access

- LDR r2, [r0, r1, LSL #2] : r2 = [r0 + (r1 * 4)]



Usage of addressing modes – sequential access

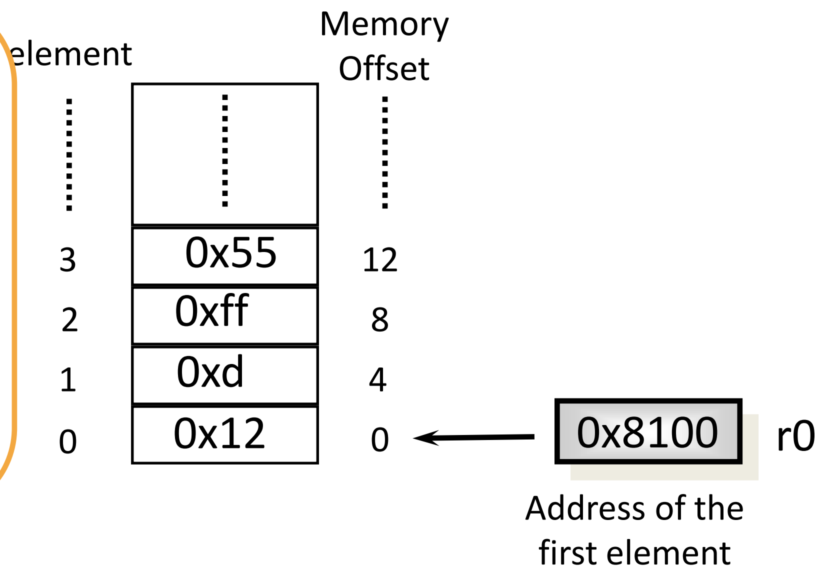
- LDR r2, [r0], #4 :
  - r2 = [r0]
  - r0 = r0 + 4
- Do we still have the address of the first item at the end?
- When do we stop?



Load and store example

- Assume an array of 4 items is saved in the memory. The first element is in the address 0x8100.
- Write an assembly code to sum the elements of this array and save the sum in the address 0x8110.

The sum should be stored just after the last item. The address of this location is 0x8110 i.e. 4 words away from the first address. That is 16 bytes away; this why we added 0x1 to the base address 0x8100.



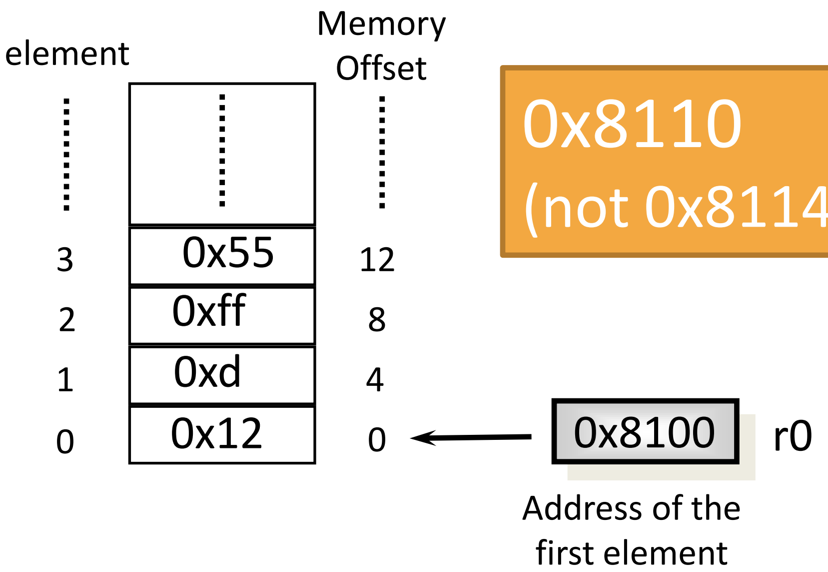
Load and store example - loop

- Design a loop to iterate four times.

```
MOV r1,#4
_loop:
@ Code in here will be repeated four times
SUBS r1,r1,#1
BNE _loop
_end: B _end
```

Load and store example – data access

```
MOV r1,#4
MOV r3,#0
LDR r0,=_data
_loop:
LDR r2, [r0], #4
ADD r3, r3, r2
SUBS r1,r1,#1
BNE _loop
STR r3, [r0]
_end: B _end
```



0x8110
(not 0x8114)

Load and store example - code

```
.section .text
.align 2
.global _start
_start:
MOV r1,#4
MOV r3,#0
LDR r0,=_data
_loop:
LDR r2, [r0], #4
ADD r3, r3, r2
SUBS r1,r1,#1
BNE _loop
STR r3, [r0]
_end: B _end

.section .data

_data: .word 0x12,0xd,0xff,0x55
```

ARM memory access instructions

LDR/STR

- Load constant and base register
- Addressing modes
- Array access

LDM/STM

- Block data transfer
- Stack

Multiple registers data transfer

- Syntax:
  - <LDM|STM>{<cond>} Rd, <address>
- Operation
  - LDM / STM allow between 1 and 16 register to be transferred to or from memory.

LDM/STM

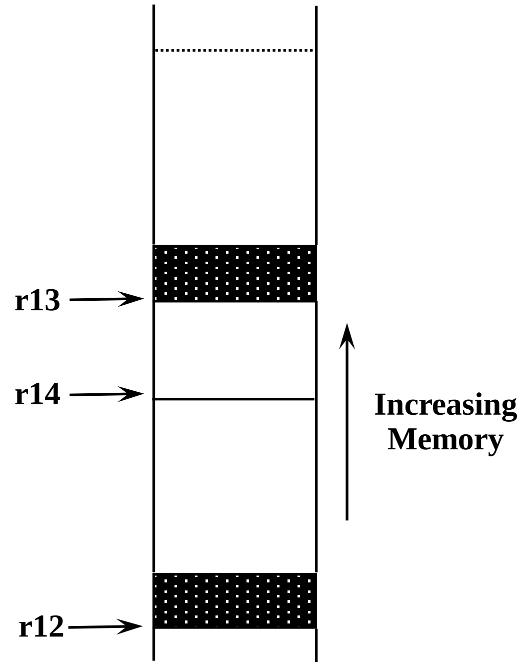
- Block data transfer
- Stack

Direct functionality of block data transfer

- STMIA / LDMIA : Increment After
- STMIB / LDMIB : Increment Before
- STMDA / LDMDA : Decrement After
- STMDB / LDMDB : Decrement Before

Example: block copy – 1/2

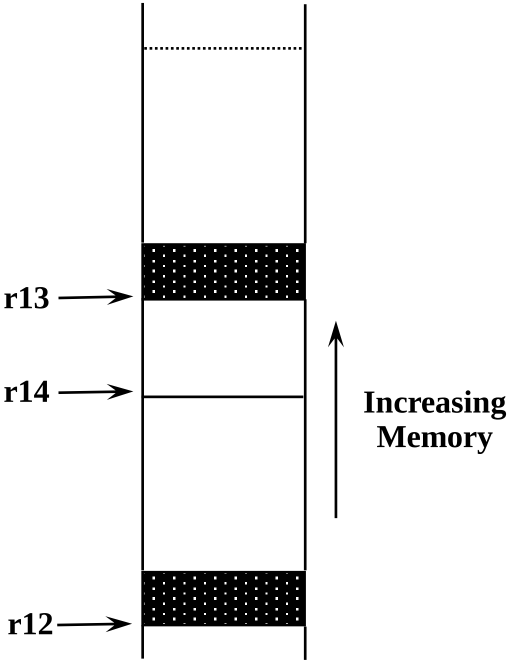
- Copy a block of memory, which is an exact multiple of 12 words long from the location pointed to by r12 to the location pointed to by r13. r14 points to the end of block to be copied.



Example: block copy – 2/2

- 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

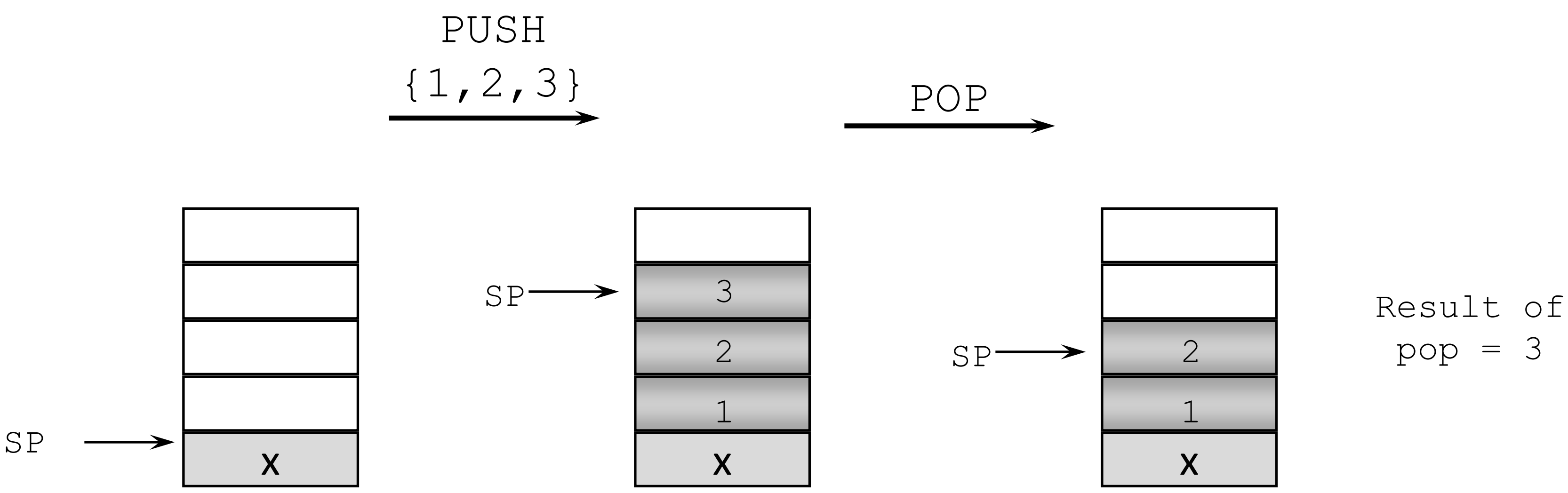
```
_loop:
LDMIA r12!, {r0-r11} @ load 48 bytes
STMIA r13!, {r0-r11} @ and store them
CMP   r12, r14       @ check for the end
BNE   _loop:    @ and loop until done
```



LDM/STM

- Block data transfer
- Stack

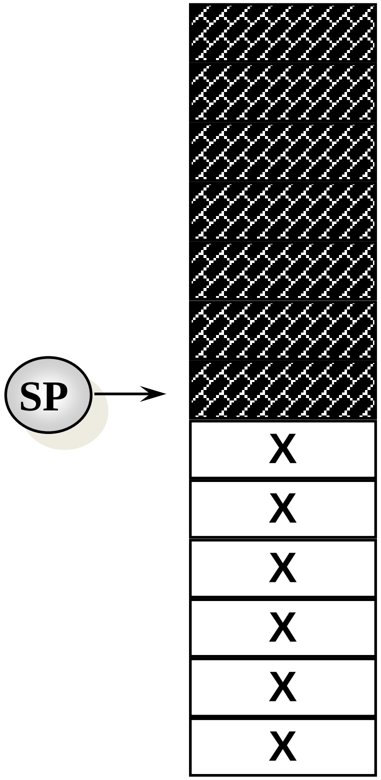
Stack



Stacks in ARM

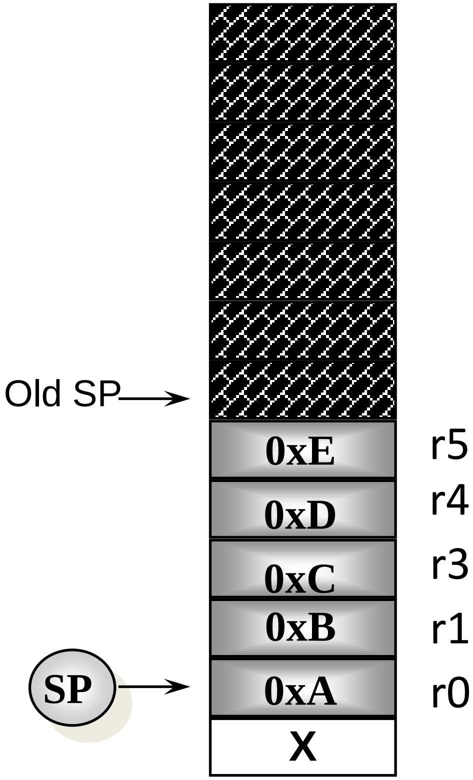
- The stack type to be used is given by the postfix to the instruction:
  - STMFD / LDMFD : Full Descending stack
  - STMFA / LDMFA : Full Ascending stack.
  - STMED / LDMED : Empty Descending stack
  - STMEA / LDMEA : Empty Ascending stack

Full Descending stack – 1/3



Register	Value
r0	0xA
r1	0xB
r3	0xC
r4	0xD
r5	0xE

```
STMFD sp!,
{r0,r1,r3-r5}
```

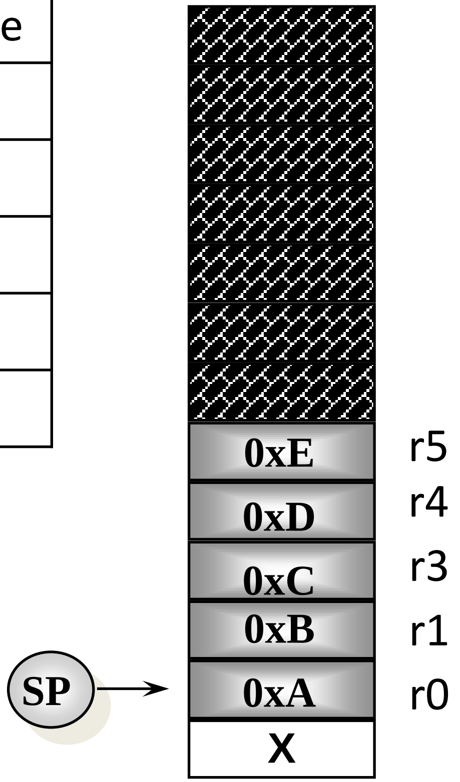


Full Descending stack – 2/3

Register	Value
r0	0xA
r1	0xB
r3	0xC
r4	0xD
r5	0xE

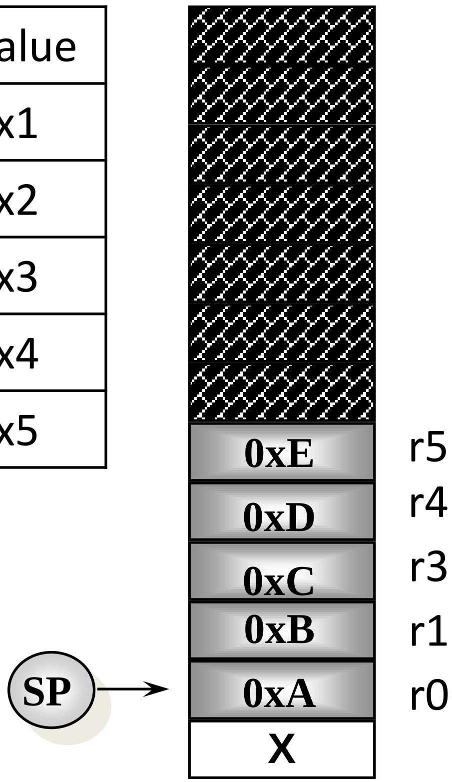
Some assembly code

Register	Value
r0	0x1
r1	0x2
r3	0x3
r4	0x4
r5	0x5

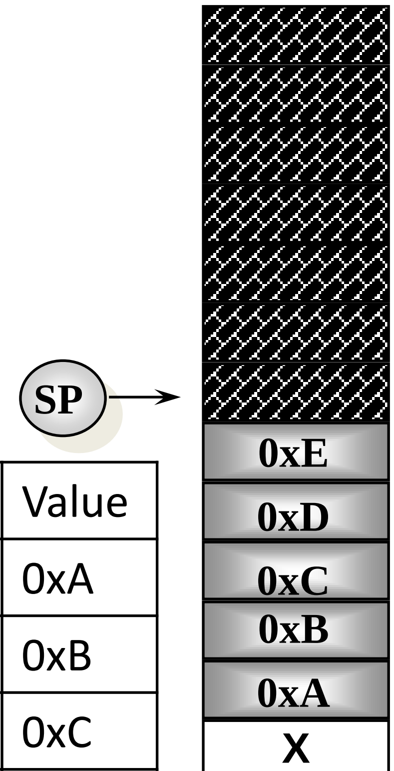


Full Descending stack – 3/3

Register	Value
r0	0x1
r1	0x2
r3	0x3
r4	0x4
r5	0x5

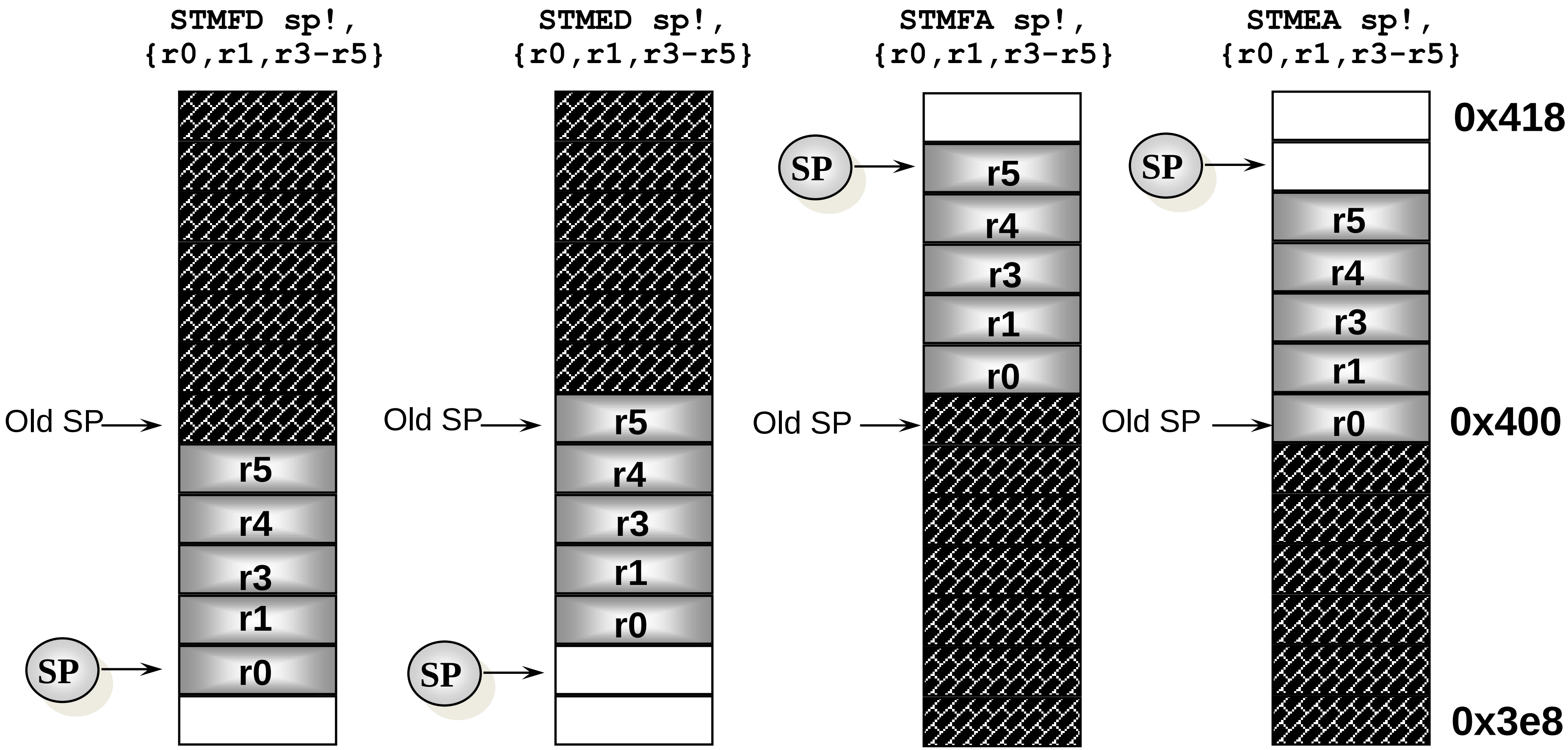



```
LDMFD sp!,
{r0,r1,r3-r5}
```



Register	Value
r0	0xA
r1	0xB
r3	0xC
r4	0xD
r5	0xE

ARM Stack Implementations



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 registers and the return address
```

………………

```
LDMFD sp!,{r0-r12, pc}
@ load all the registers and return
  automatically
```

Summary

- LDR/STR
  - Load constant and base register
  - Addressing modes
  - Array access
- LDM/STM
  - Block data transfer
  - Stack