Advanced Micro-controllers - ARM

DESD @ Sunbeam Infotech

Instruction Set basics

- The ARM Architecture is a Load/Store architecture
  - No direct manipulation of memory contents
  - Memory must be loaded into the CPU to be modified, then written back out

- Cores are either in ARM *state* or Thumb *state*
  - This determines which instruction set is being executed
  - An instruction must be executed to switch between states

- The architecture allows programmers and compilation tools to reduce branching through the use of conditional execution
  - Method differs between ARM and Thumb, but the principle is that most (ARM) or all (Thumb) instructions can be executed conditionally.

ARM instruction set

- Data transfer instructions
- Arithmetic instructions
- Logical instructions
- Conditional branching and if-then instruction
- Barrel shifter
- Load-Store instructions
  - Post increment
  - Pre increment
  - Pre increment + Inline barrel shifter
  - Pre increment with write-back
- Load-Store multiple
- Function call and stack operations
- DSP instructions
- Miscellaneous instructions

Data transfer instructions

- mov rd, #K rd = K # → 'immediate value'
- mvn rd, #K rd = ~K
- mrs rd, special_register
- msr special_register, rs

Arithmetic instructions

- add rd, rm, rn rd = rm + rn
- add rd, rs rd = rd + rs
- sub rd, rm, rn rd = rm - rn
- subs rd, rm, rn rd = rm - rn → update NZCV in xPSR
- mul rd, rm, rn rd = rm * rn
- mla rd, rm, rn, ra rd = rm * rn + ra (DSP instn)
- udiv rd, rn, rm rd = rn ÷ rm

Logical instructions

- and rd, rn, rm rd = rn & rm
- bic rd, rn, rm rd = rn & ~rm
- tst rm, rn rn & rm → update NZCV flags
- orr rd, rn, rm rd = rn | rm
- orn rd, rn, rm rd = rn | ~rm
- eor rd, rn, rm rd = rn ^ rm

Conditional branching

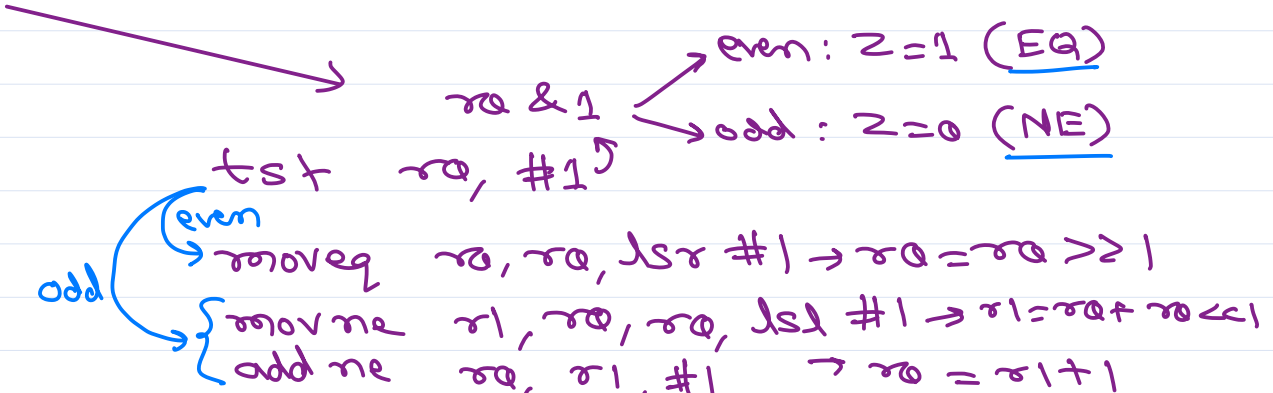
- cmp rm, rn

rm - rn → update NZCV flags
if Z = 1, then nums are equal
if Z = 0, then nums are not equal

- bxx label
  - Branching based on condition

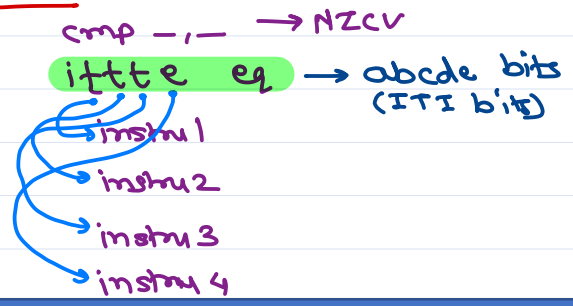
beq, bne, bgt, blt, bge, ble ... } internally check NZCV flags & jump accordingly.
↓
Z

- Conditional execution of ARM instructions (not thumb)
  - movxx rd, rs
  - addxx rd, rm, rn



- Thumb-2 if-then instruction
  - cmp rm, rn
  - ite gt
  - movgt rd, rm
  - movle rd, rn

or tst → NZCV flags



Barrel shifter

LSL : Logical Left Shift

Multiplication by a power of 2

ASR: Arithmetic Right Shift

Division by a power of 2, preserving the sign bit

LSR : Logical Shift Right

Division by a power of 2

ROR: Rotate Right

Bit rotate with wrap around from LSB to MSB

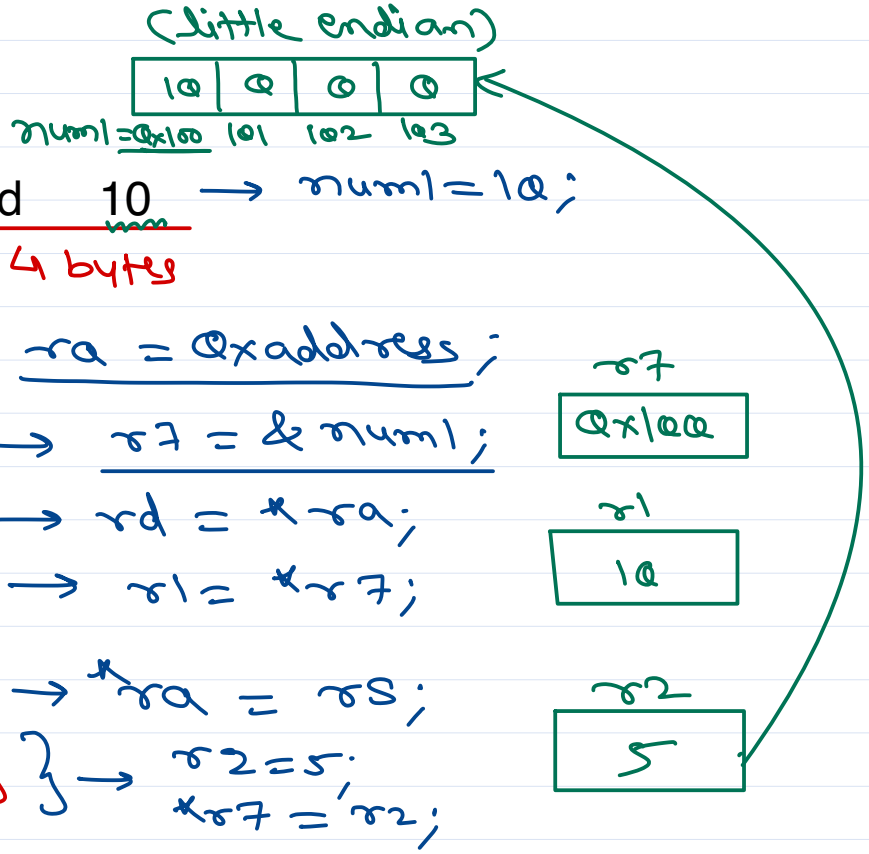
RRX: Rotate Right Extended

Single bit rotate with wrap around from CF to MSB

- Inline barrel shifter
  - mov rd, rs, lsl #k
  - mov rd, rs, lsr #k

Load store instructions

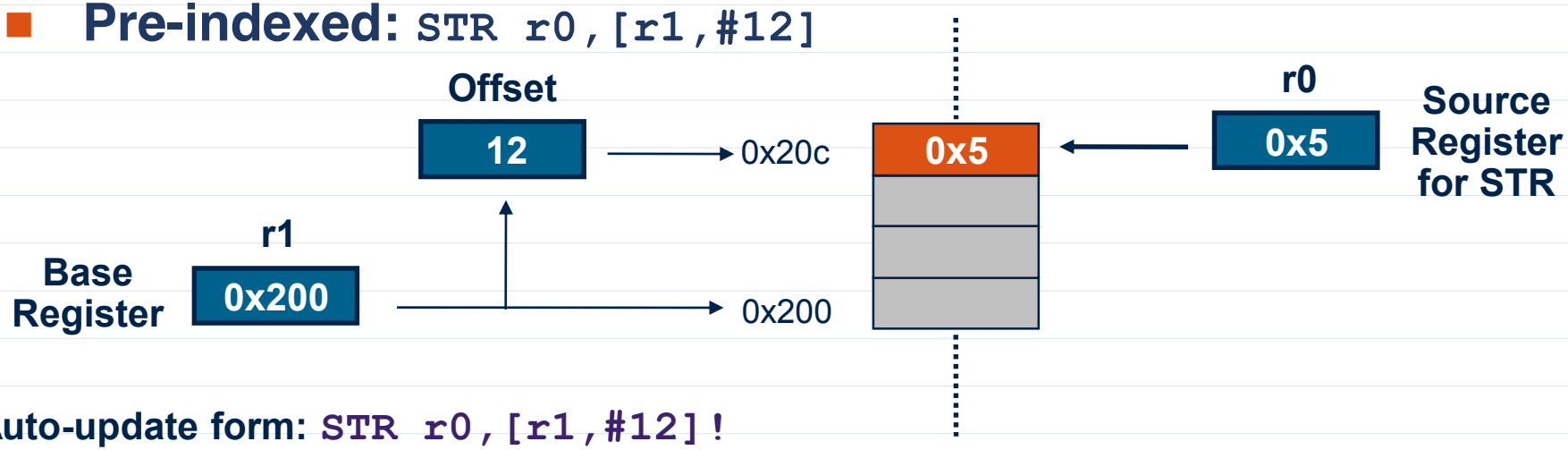
- Global variables
  - .section .data
  - num1: .word 10



- ldr ra, =addr
  - e.g. ldr r7, =num1 → r7 = &num1;
  - const value
  - ra = 0xaddress;
- ldr rd, [ra]
  - e.g. ldr r1, [r7] → r1 = *r7;
  - → rd = *ra;
- str rs, [ra]
  - e.g. mov r2, #5 / str r2, [r7] → r2=5; *r7 = r2;
  - → *ra = rs;
- ldrb, ldrh, ldr.
- ldrsb, ldrsh

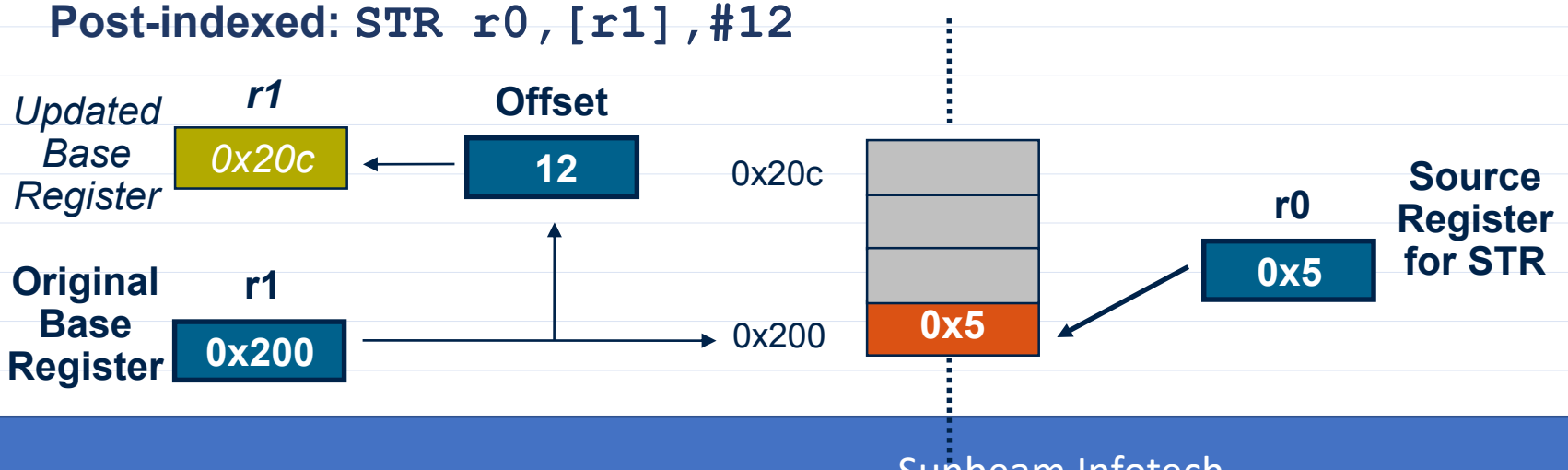
Load/Store Pre/Post-increment

- **Pre-indexed:** `STR r0,[r1,#12]`



Auto-update form: `STR r0,[r1,#12]!`

- **Post-indexed:** `STR r0,[r1],#12`



Load/Store Multiple

- Syntax:

  <LDM|STM>{<cond>}<addressing_mode> Rb{!}, <register list>

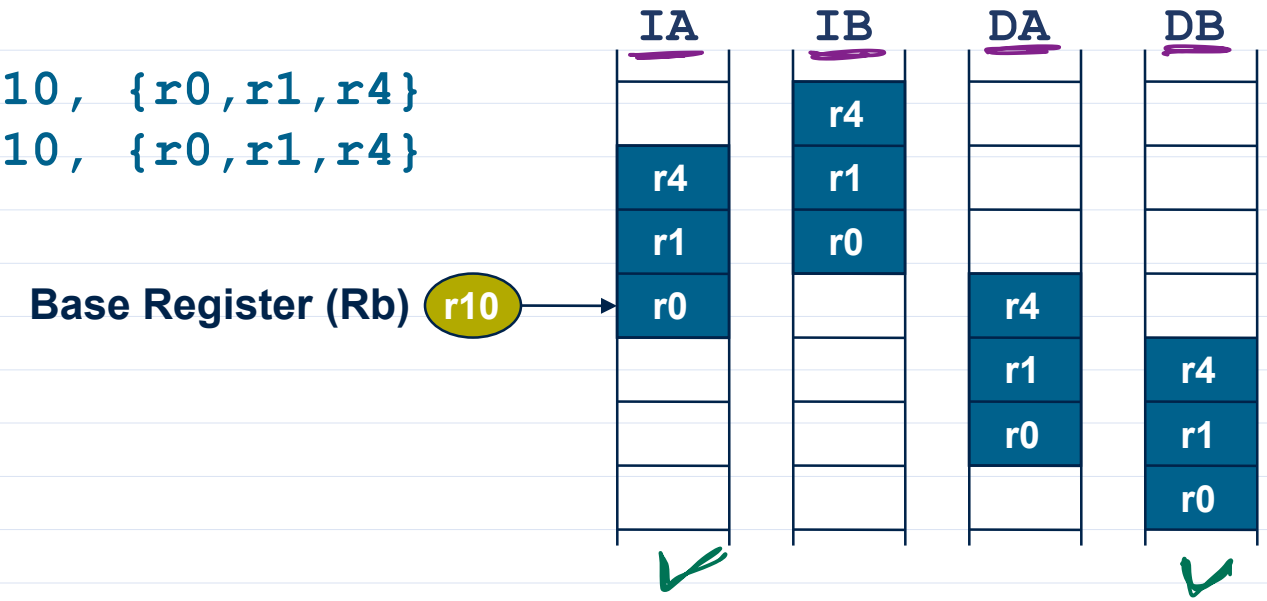
- 4 addressing modes:

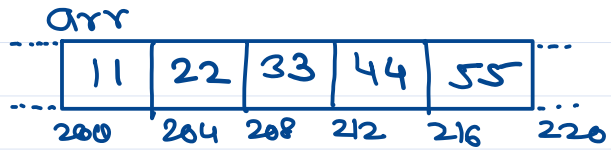
default in Cortex-M

LDMIA / STMIA	increment after
~~LDMIB / STMIB~~	~~increment before~~
~~LDMDA / STMDA~~	~~decrement after~~
LDMDB / STMDB	decrement before

Not supported in Cortex-M

```
LDMxx r10, {r0,r1,r4}
STMxx r10, {r0,r1,r4}
```

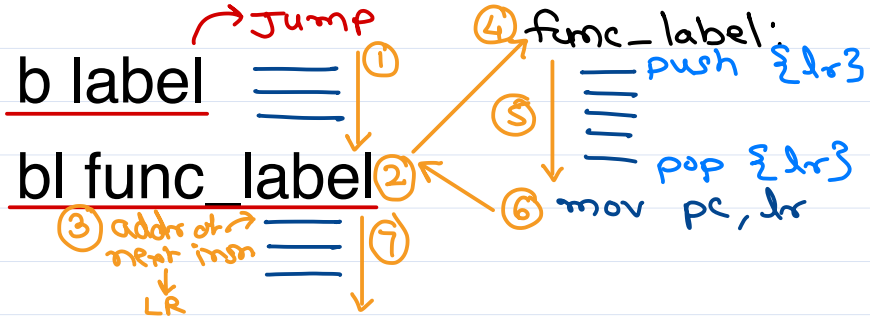





```
ldr     r7, =arr
ldmia   r7, {r1-r5}
ldmia   r7!, {r1-r5}
ldr     r7, =arr
add     r7, #20
ldmdb   r7, {r1-r5}
ldmdb   r7!, {r1-r5}
```

Cortex M support only IA & DB.

Function call and stack operations



- b label
- bl func_label
- stmfd sp!, {lr}
- push {lr}

} Store lr on stack

- mov pc, lr

- ldmfd sp!, {pc}
- pop {pc}

} Pop return addr into pc from stack

- AAPCS
  - Arguments: r0, r1, r2, r3
  - Return value: r0

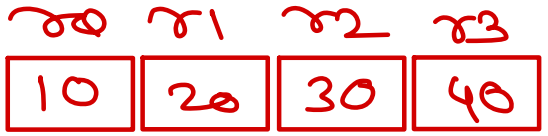
arg1 arg2 arg3 arg4 arg5..... pushed on stack

  - Called saved: r4-r11, lr
  - Caller saved: r0-r3, r12

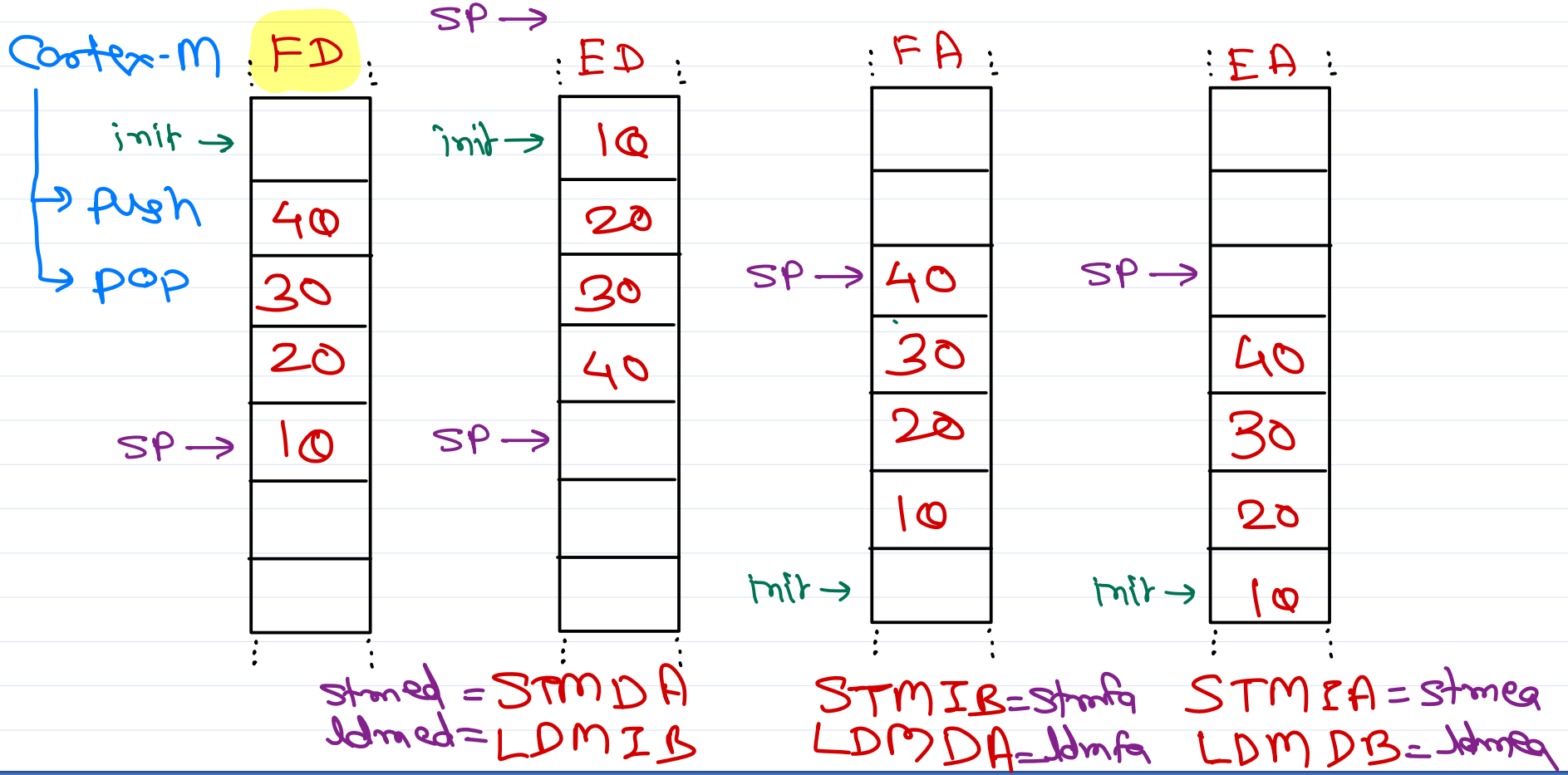
(4 bytes) ldr or str → sp
(2 bytes) ldrh or strh
(1 byte) ldrb or strb
(8 bytes) ldrd or strd

expect addr to be dword aligned (multiple of 8)

Stack types



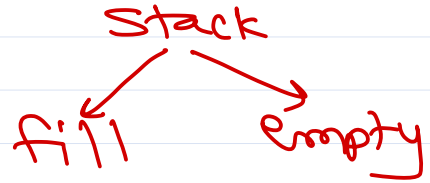
stmdb sp!, {r0-r3} = stmfd
ldmia sp!, {r0-r3} = ldmfd

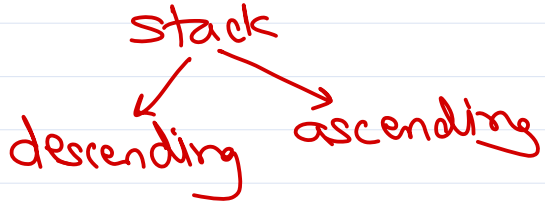


stmed = STMDA
ldmed = LDMIB

STMIB = stmfa
LDMDA = ldmfa

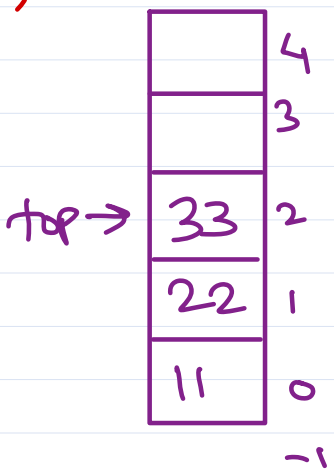
STMIA = stmea
LDMDB = ldmea



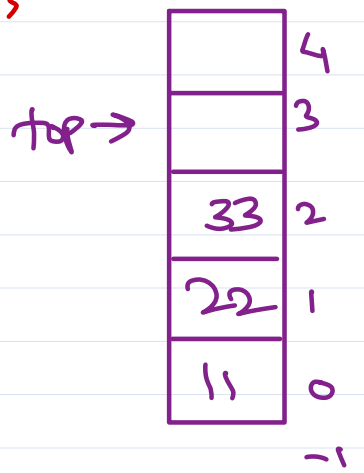


Stack in Data Structures

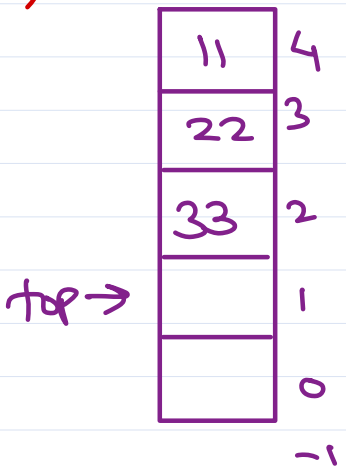
```
arr[5]
top = -1;

push(val) {
    top++;
    arr[top] = val;
}

pop() {
    val = arr[top];
    top--;
}
```



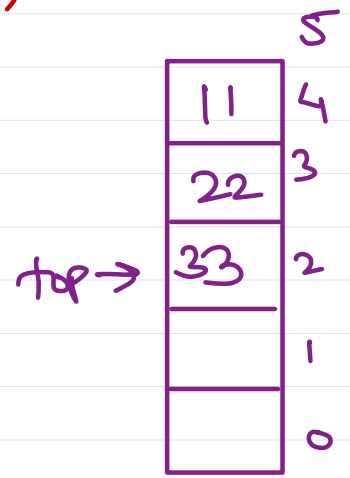

```
arr[5]
top = 0;

push(val) {
    arr[top] = val;
    top++;
}

pop() {
    top--;
    val = arr[top];
}
```




```
arr[5]
top = 4;

push(val) {
    arr[top] = val;
    top--;
}

pop() {
    top++;
    val = arr[top];
}
```

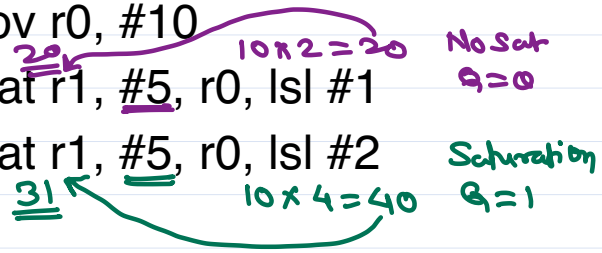



```
arr[5]
top = 5;

push(val) {
    top--;
    arr[top] = val;
}

pop() {
    val = arr[top];
    top++;
}
```



DSP instructions

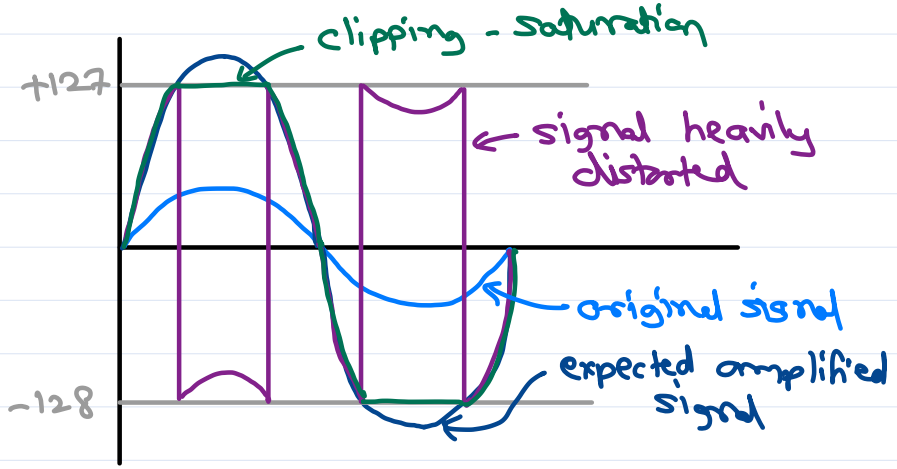
- Saturated Math
  - mov r0, #10
  - usat r1, #5, r0, lsl #1
  - usat r1, #5, r0, lsl #2



@ r1 = MIN(r0 * 2, 31) --> 20 (q=0)

@ r1 = MIN(r0 * 4, 31) --> 31 (q=1)

- SIMD instructions
  - ldr r1, =0x11223344
  - ldr r2, =0x44332211
  - qadd8 r0, r1, r2



Miscellaneous instructions

- rev instruction
  - ldr r0, =0x11223344
  - rev r1, r0 @ r1 -- 0x44332211
- sign extend
  - ldr r0, =0x55AA8765
  - sxtb r1, r0 @ last byte of r0 -- 0110 0101 @ new value of r1 will be -- 0x00000065
  - sxth r2, r0 @ last 2 bytes of r0 -- 1000 0111 0110 0101 @ new value of r2 will be -- 0xffff8765
  - uxth r3, r0 @ new value of r3 will be -- 0x00008765
- bit-field extrac
  - ldr r6, =0x11223344 @ assume value of ADGDR is 0x11223344
  - ubfx r0, r6, #4, #12 @ new val of r6 will be = 0x0334
- clear/insert bits
  - ldr r1, =0x11223344
  - bfc r1, #8, #16 @ r1 will be 0x11000044
  - mov r0, 0x12
  - bfi r1, r0, #8, #16 @ r1 will be 0x11001244

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>