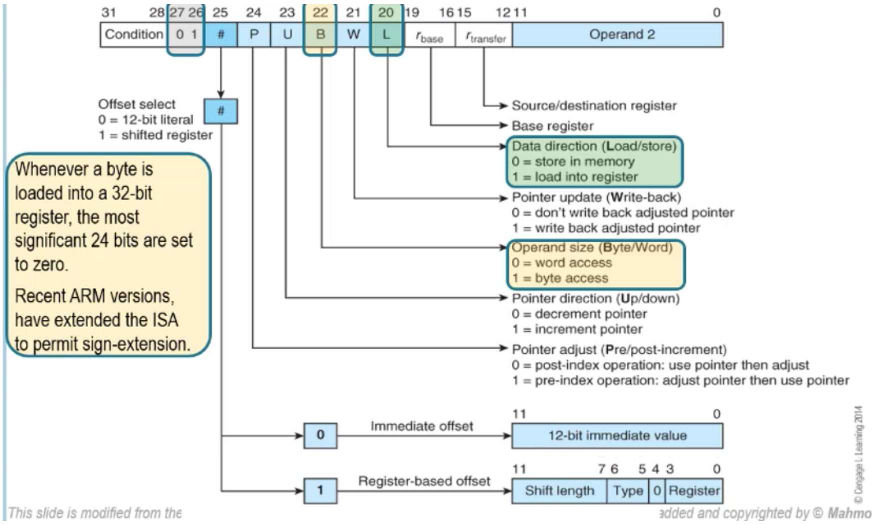
Week 11

LDR and STR instructions:



Offset select
0 = 12-bit literal
1 = shifted register

Whenever a byte is loaded into a 32-bit register, the most significant 24 bits are set to zero.

Recent ARM versions, have extended the ISA to permit sign-extension.

Source/destination register

Base register

Data direction (**L**oad/store)
0 = store in memory
1 = load into register

Pointer update (**W**rite-back)
0 = don't write back adjusted pointer
1 = write back adjusted pointer

Operand size (**B**yte/Word)
0 = word access
1 = byte access

Pointer direction (**U**p/down)
0 = decrement pointer
1 = increment pointer

Pointer adjust (**P**re/post-increment)
0 = post-index operation: use pointer then adjust
1 = pre-index operation: adjust pointer then use pointer

Immediate offset

12-bit immediate value

Register-based offset

This slide is modified from the

added and copyrighted by © Mahmo

Stack Data Structure

- Items enter at one end and leave from the same end
- Last item you put in is the first one you take out
- Implemented using a stack pointer to point to top of the stack (TOS)
  - As items are pushed onto the stack, stack pointer decreases
  - As items are removed, stack pointer increases

4 Ways of Constructing a Stack

1. **Stack grows up, stack pointer points to TOS**

   - Each stack entry is 4 bytes
   - To push an entry, we subtract the pointer by 4 and store the new data in this stack pointer
     - STR R0, [SP, #-4]!
   - To pop an entry, we pull data off the stack and increase the stack pointer by 4
     - LDR R0, [SP], #4

2. **Grows up but stack pointer points to the first free space**

   - To push an entry, we store data in the location of the stack pointer and then decrease the stack pointer by 4
     - STR R0, [SP], #-4
   - To pop an entry, we increase stack pointer and then pull data off the stack
     - LDR R0, [SP, #4]!

3. **Stack grows down, stack pointer points to TOS**

   - To push an entry, we increase the stack pointer by 4 and push data onto the stack
     - STR R0, [SP, #4]!
   - To pop an entry, we pull data from stack and subtract stack pointer by 4
     - LDR R0, [SP], #-4

4. **Stack grows down, stack pointer points to next free space**

   - To push an entry, we push data onto the stack and increase stack pointer by 4
     - STR R0, [SP], #4
   - To pop an entry, we decrement the stack pointer by 4 and then pull data from stack
     - LDR R0, [SP, #-4]!

Two decisions to determine what type of stack you need:

1. Whether the stack grows up or down?
2. Whether stack pointer points to TOS or first free empty space

CISC processors automatically maintain the stack.
RISC (ARM) processors force programmers to maintain the stack.