COL788: Advanced Topics in Embedded Computing

Lecture 8 – Processor Architecture (Cont.)

Vireshwar Kumar
CSE@IITD

August 25, 2022

Semester I
2022-2023

Last Lecture

- Arithmetic Operations
- Comparisons
- Logical Operations
- Data Movement
- Barrel Shifter

Immediate Value

- There is no single instruction which will load a 32-bit constant into a register without performing a data load from memory.
  - All ARM instructions are 32 bits long
  - ARM instructions do not use the instruction stream as data.
- The data processing instruction format has 12 bits available for operand2
  - If used directly this would only give a range of 4096.
- Instead it is used to store 8 bit constants, giving a range of 0 - 255.
- These 8 bits can then be rotated right through an even number of positions (ie RORs by 0, 2, 4,..30).
  - This gives a much larger range of constants that can be directly loaded, though some constants will still need to be loaded from memory.

Range of Immediate Value

- This gives us:
  - 0 - 255 [0 - 0xff]
  - 256,260,264,..,1020 [0x100-0x3fc, step 4, 0x40-0xff ror 30]
  - 1024,1040,1056,..,4080 [0x400-0xff0, step 16, 0x40-0xff ror 28]
  - 4096,4160, 4224,..,16320 [0x1000-0x3fc0, step 64, 0x40-0xff ror 26]
- Example:
  - MOV r0, #0x40, 26 => MOV r0, #0x1000

Multiplication using a Shifted Register

- Using a multiplication instruction to multiply by a constant means first loading the constant into a register and then waiting a number of internal cycles for the instruction to complete.
- A more optimum solution can often be found by using some combination of MOVs, ADDs, SUBs and RSBs with shifts.
  - Multiplications by a constant equal to a ((power of 2) ± 1) can be done in one cycle.
- Example:
  - r0 = r1 * 5 = r1 + (r1 * 4)
  - ADD r0, r1, r1, LSL #2
- Example:
  - r2 = r3 * 105 = r3 * 15 * 7 = r3 * (16 - 1) * (8 - 1)
  - RSB r2, r3, r3, LSL #4 ; r2 = r3 * 15
  - RSB r2, r2, r2, LSL #3 ; r2 = r2 * 7

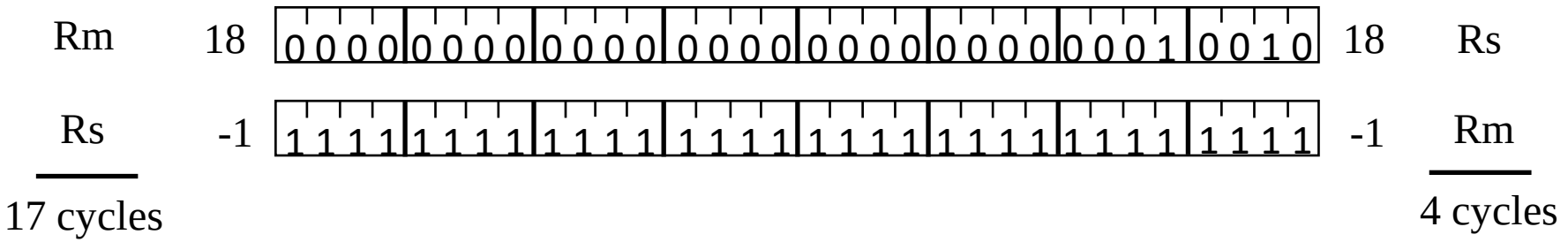
Multiplication Instructions

- The Basic ARM provides two multiplication instructions.
- Multiply
  - MUL{<cond>}{S} Rd, Rm, Rs ; Rd = Rm * Rs
- Multiply Accumulate - does addition for free
  - MLA{<cond>}{S} Rd, Rm, Rs,Rn ; Rd = (Rm * Rs) + Rn
- Restrictions on use:
  - Rd and Rm cannot be the same register
    - Can be avoid by swapping Rm and Rs around. This works because multiplication is commutative.
  - Cannot use program counter

  These will be picked up by the assembler if overlooked.
- Operands can be considered signed or unsigned
  - Up to user to interpret correctly.

Multiplication Implementation

- The ARM makes use of Booth's Algorithm to perform integer multiplication.
- On some ARMs this operates on 2 bits of Rs at a time.
  - For each pair of bits this takes 1 cycle (plus 1 cycle to start with).
  - However when there are no more 1's left in Rs, the multiplication will early-terminate.
- Example: Multiply 18 and -1 : Rd = Rm * Rs



Loading 32-bit Constants

- Although the MOV/MVN mechanism will load a large range of constants into a register, sometimes this mechanism will not generate the required constant.
- Therefore, the assembler also provides a method which will load any 32-bit constant:
  - `LDR rd,=numeric constant`
- If the constant can be constructed using either a MOV or MVN then this will be the instruction actually generated.
- Otherwise, the assembler will produce an LDR instruction with a PC-relative address to read the constant from a literal pool.
  - `LDR r0,=0x42 ; generates MOV r0,#0x42`
  - `LDR r0,=0x55555555 ; generate LDR r0,[pc, offset to lit pool]`
- As this mechanism will always generate the best instruction for a given case, it is the recommended way of loading constants.

Load / Store Instructions

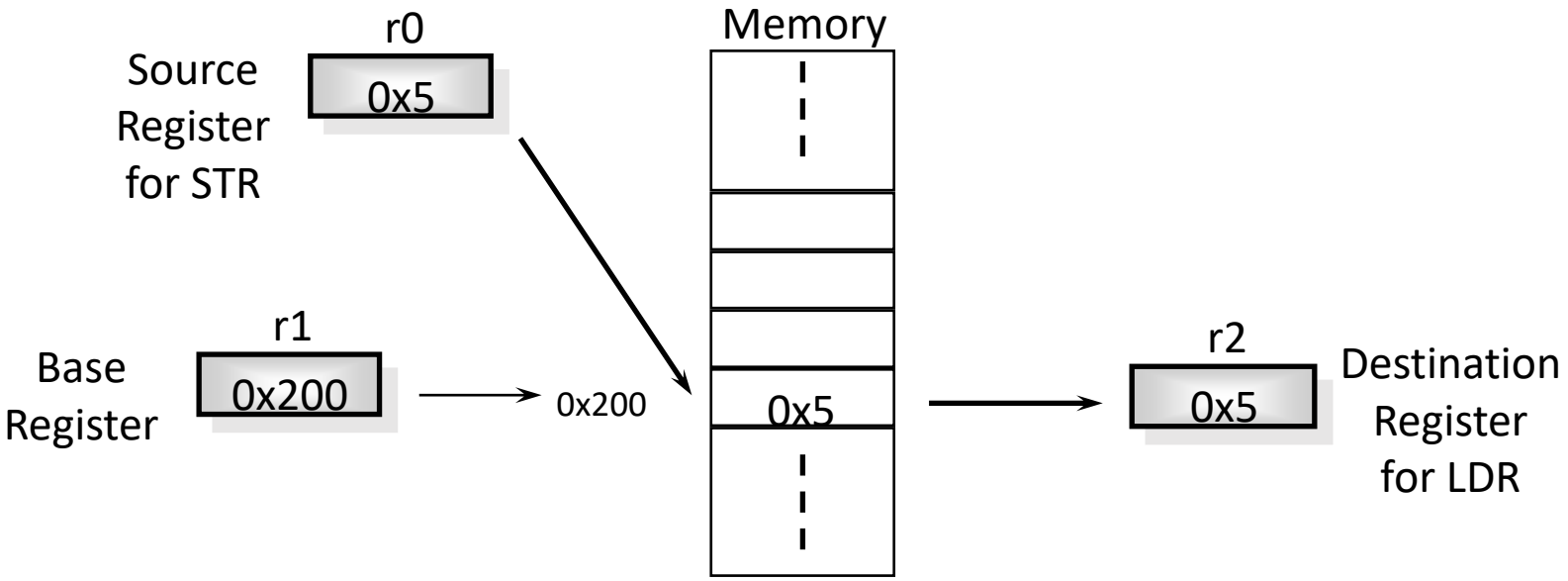
- The ARM is a Load / Store Architecture:
  - Does not support memory to memory data processing operations.
  - Must move data values into registers before using them.
- This might sound inefficient, but in practice isn’t:
  - Load data values from memory into registers.
  - Process data in registers using a number of data processing instructions which are not slowed down by memory access.
  - Store results from registers out to memory.
- The ARM has three sets of instructions which interact with main memory. These are:
  - Single register data transfer (LDR / STR).
  - Block data transfer (LDM/STM).
  - Single Data Swap (SWP).

Single Register Data Transfer

- The basic load and store instructions are:
  - Load and Store Word or Byte
    - LDR / STR / LDRB / STRB
- ARM Architecture Version 4 also adds support for halfwords and signed data.
  - Load and Store Halfword
    - LDRH / STRH
  - Load Signed Byte or Halfword - load value and sign extend it to 32 bits.
    - LDRSB / LDRSH
- All of these instructions can be conditionally executed by inserting the appropriate condition code after STR / LDR.
  - e.g. LDREQB
- Syntax:
  - <LDR|STR>{<cond>}{<size>} Rd, <address>

Load and Store: Base Register

- The memory location to be accessed is held in a base register
  - STR r0, [r1] ; Store contents of r0 to location pointed to ; by contents of r1.
  - LDR r2, [r1] ; Load r2 with contents of memory location ; pointed to by contents of r1.

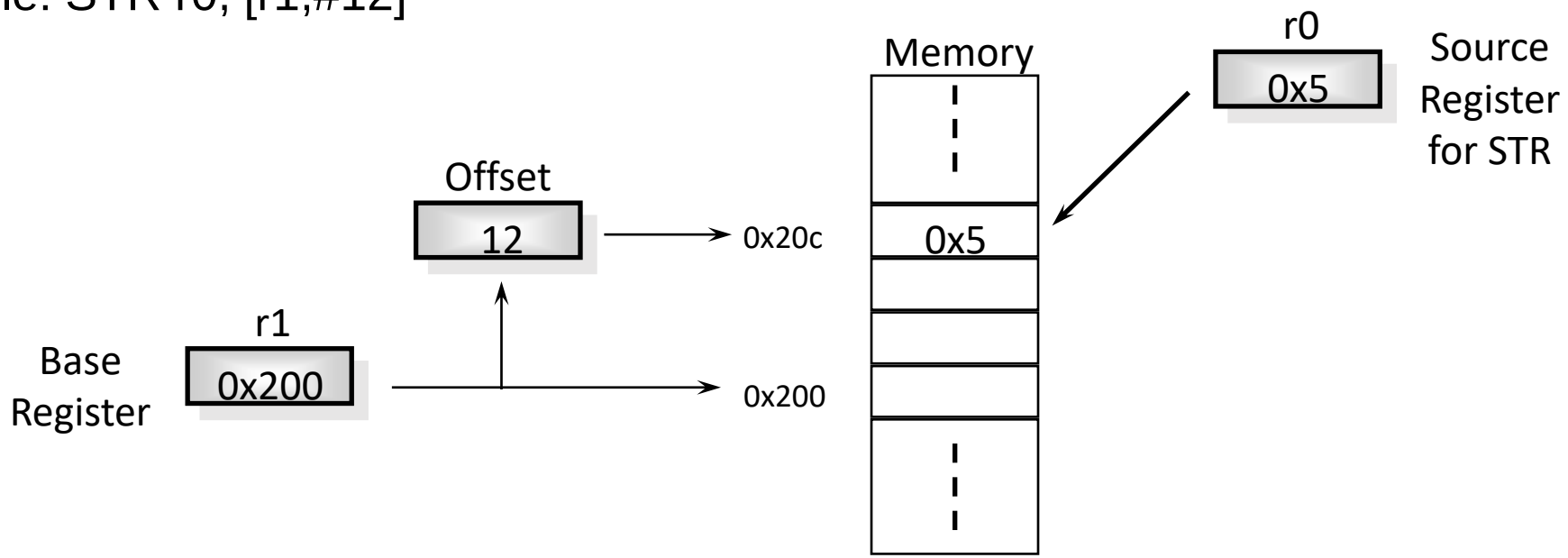


Load and Store: Offsets from Base Register

- As well as accessing the actual location contained in the base register, these instructions can access a location offset from the base register pointer.
- This offset can be
  - An unsigned 12bit immediate value (ie 0 - 4095 bytes).
  - A register, optionally shifted by an immediate value
- This can be either added or subtracted from the base register:
  - Prefix the offset value or register with '**+**' (default) or '**-**'.
- This offset can be applied:
  - before the transfer is made: ***Pre-indexed addressing***
    - optionally *auto-incrementing* the base register, by postfixing the instruction with an '**!**'.
  - after the transfer is made: ***Post-indexed addressing***
    - causing the base register to be *auto-incremented*.

Load and Store: Pre-indexed Addressing

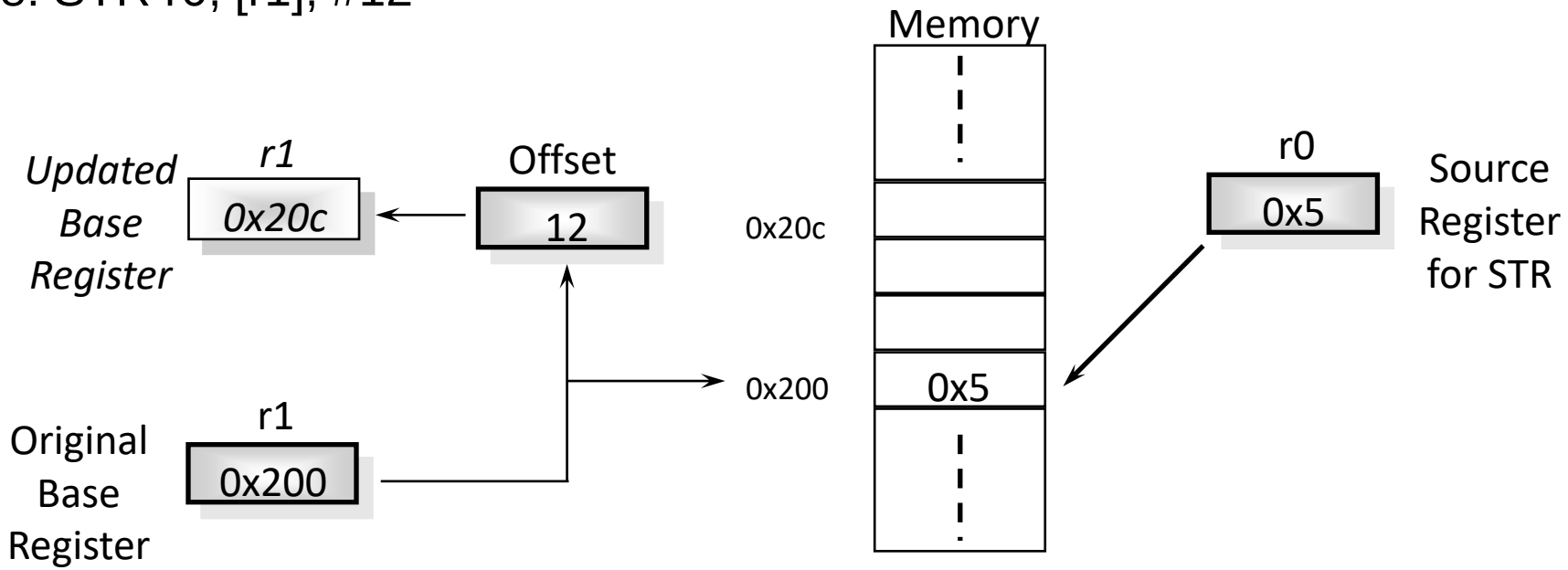
- Example: STR r0, [r1,#12]



- To store to location 0x1f4 instead use: STR r0, [r1,#-12]
- To auto-increment base pointer to 0x20c use: STR r0, [r1, #12]!
- If r2 contains 3, access 0x20c by multiplying this by 4:
  - STR r0, [r1, r2, LSL #2]

Load and Store: Post-indexed Addressing

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



- To auto-increment the base register to location 0x1f4 instead use:
  - STR r0, [r1], #-12
- If r2 contains 3, auto-incremenet base register to 0x20c by multiplying this by 4:
  - STR r0, [r1], r2, LSL #2

What’s Next?

- Next Lecture (August 29, Monday, 11 am – 12 pm)
  - Lecture 9