SNS COLLEGE OF ENGINEERING

(Autonomous)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

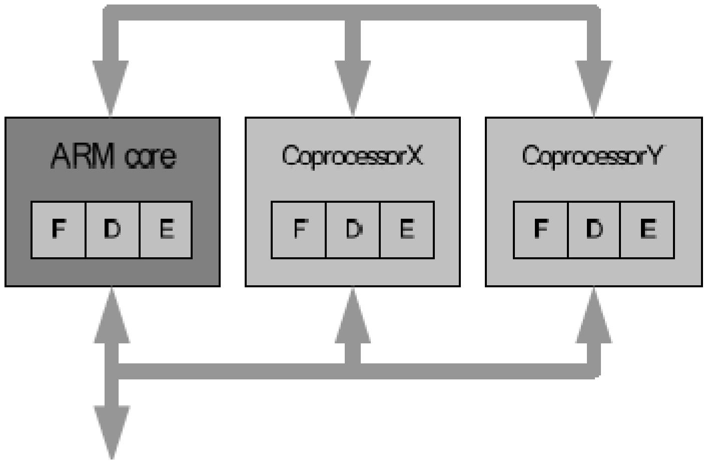
ARM INSTRUCTION SET

Main features of the ARM Instruction Set

- All instructions are 32 bits long.
- Most instructions execute in a single cycle.
- Most instructions can be conditionally executed.
- A load/store architecture
  - Data processing instructions act only on registers
    - Three operand format
    - Combined ALU and shifter for high speed bit manipulation
  - Specific memory access instructions with powerful auto-indexing addressing modes.
    - 32 bit and 8 bit data types
      - and also 16 bit data types on ARM Architecture v4.
    - Flexible multiple register load and store instructions
- Instruction set extension via coprocessors
- Very dense 16-bit compressed instruction set (Thumb)

Coprocessors

- Up to *16* coprocessors can be defined
- Expands the ARM instruction set
- Each coprocessor can have up to 16 private registers of any reasonable size
- Load-store architecture



Thumb

- Thumb is a 16-bit instruction set
  - Optimized for code density from C code
  - Improved performance form narrow memory
  - Subset of the functionality of the ARM instruction set
- Core has two execution states – ARM and Thumb
  - Switch between them using BX instruction
- Thumb has characteristic features:
  - Most Thumb instruction are executed unconditionally
  - Many Thumb data process instruction use a 2-address format
  - Thumb instruction formats are less regular than ARM instruction formats, as a result of the dense encoding.

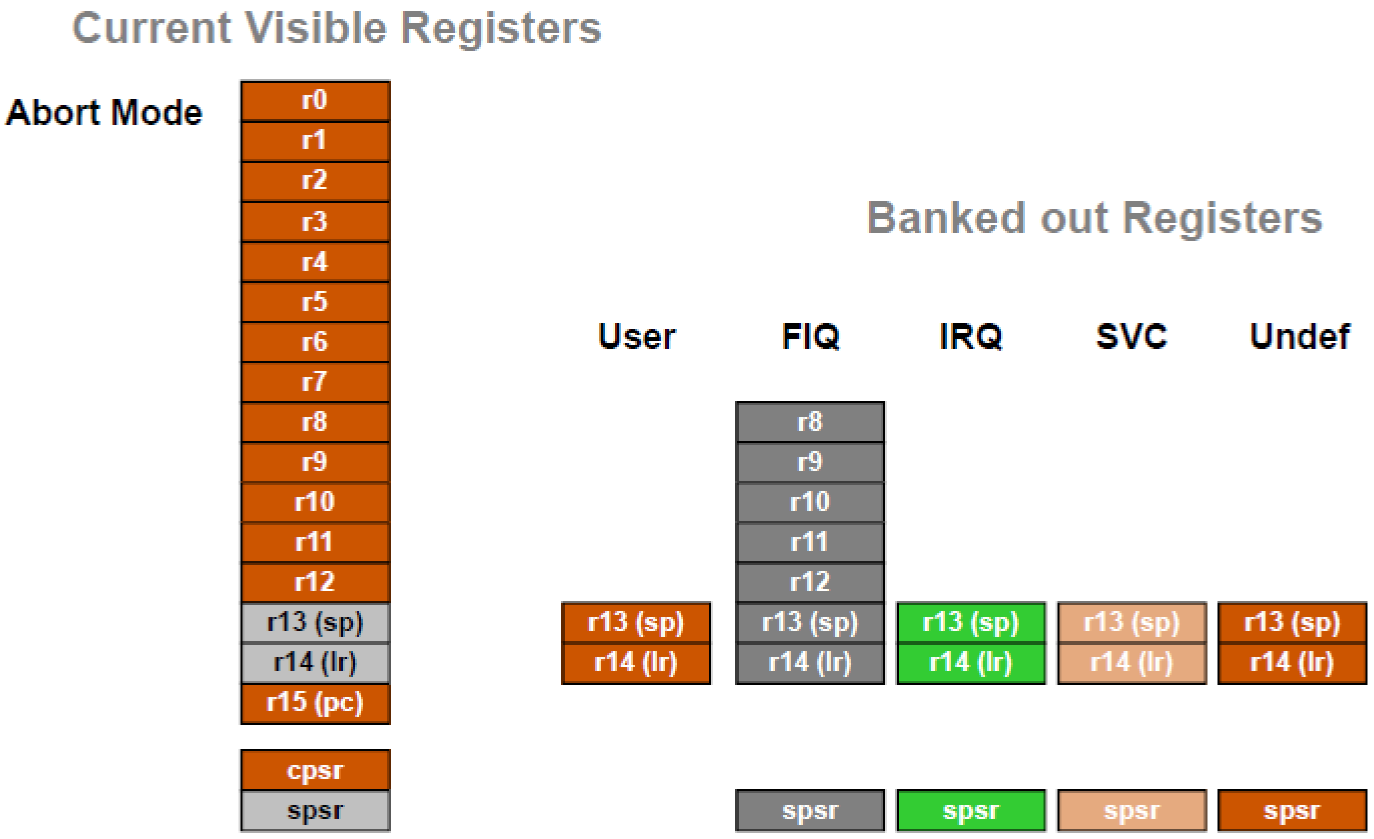
Processor Modes

- The ARM has six operating modes:
  - User (unprivileged mode under which most tasks run)
  - FIQ (entered when a high priority (fast) interrupt is raised)
  - IRQ (entered when a low priority (normal) interrupt is raised)
  - Supervisor (entered on reset and when a Software Interrupt instruction is executed)
  - Abort (used to handle memory access violations)
  - Undef (used to handle undefined instructions)
- ARM Architecture Version 4 adds a seventh mode:
  - System (privileged mode using the same registers as user mode)

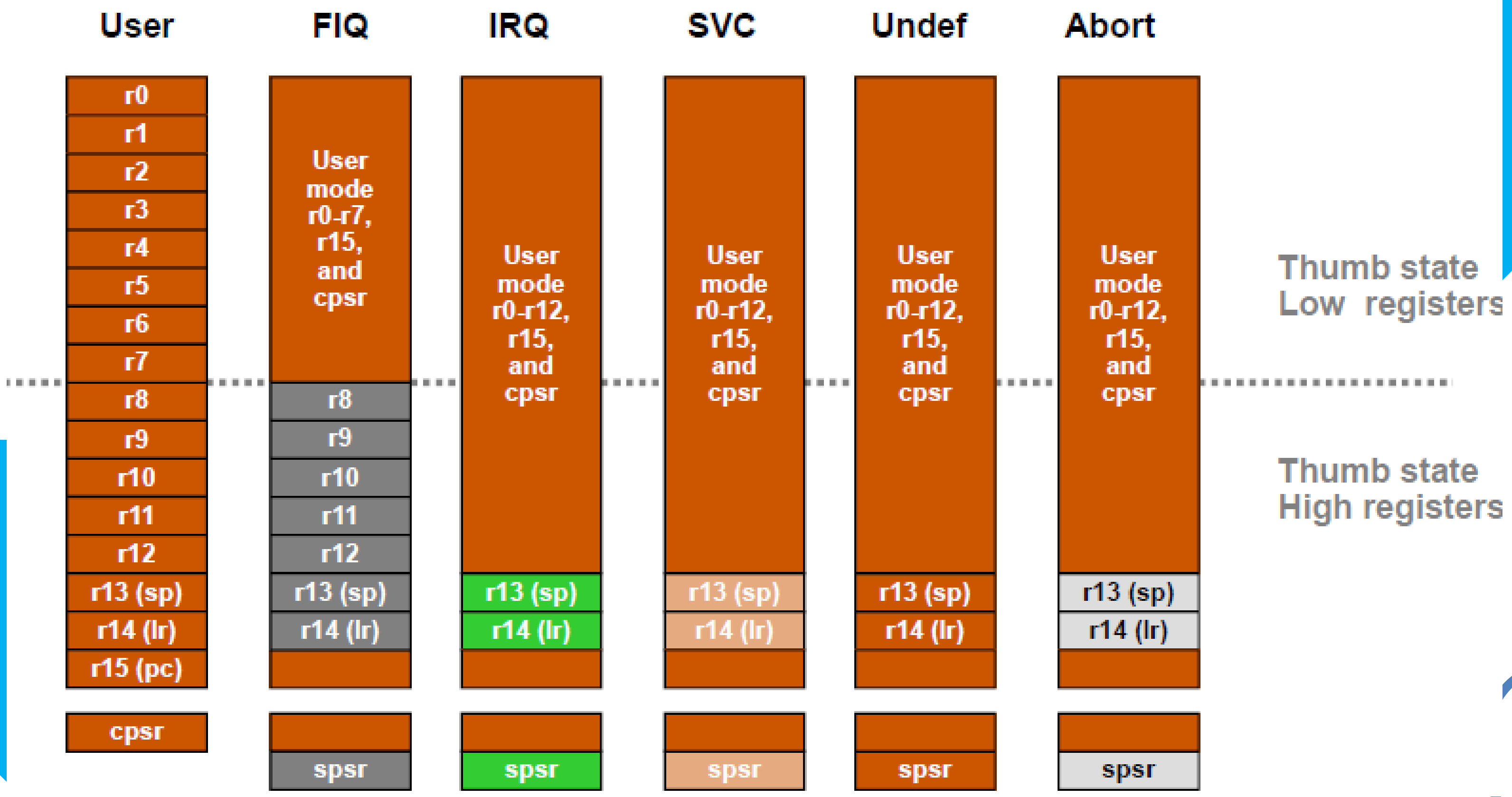
The Registers

- ARM has 37 registers in total, all of which are 32-bits long.
  - 1 dedicated program counter
  - 1 dedicated current program status register
  - 5 dedicated saved program status registers
  - 30 general purpose registers
- However these are arranged into several banks, with the accessible bank being governed by the processor mode. Each mode can access
  - a particular set of r0-r12 registers
  - a particular r13 (the stack pointer) and r14 (link register)
  - r15 (the program counter)
  - cpsr (the current program status register)
- And privileged modes can also access
  - a particular spsr (saved program status register)

The ARM Register Set



Register Organization Summary


User
FIQ
IRQ
SVC
Undef
Abort
r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 (sp)
r14 (lr)
r15 (pc)
cpsr
User mode r0-r7, r15, and cpsr
r8
r9
r10
r11
r12
r13 (sp)
r14 (lr)
spsr
User mode r0-r12, r15, and cpsr
r13 (sp)
r14 (lr)
spsr
User mode r0-r12, r15, and cpsr
r13 (sp)
r14 (lr)
spsr
User mode r0-r12, r15, and cpsr
r13 (sp)
r14 (lr)
spsr
User mode r0-r12, r15, and cpsr
r13 (sp)
r14 (lr)
spsr
Thumb state Low registers
Thumb state High registers

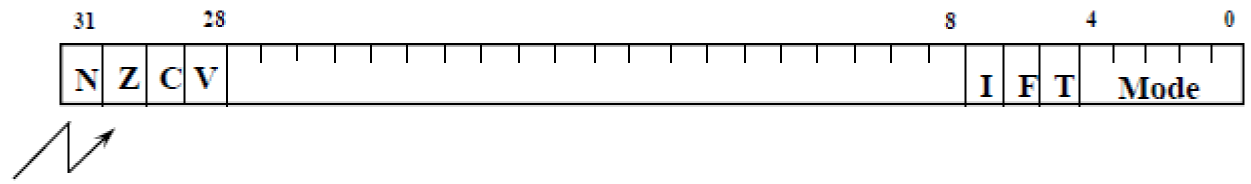
Accessing Registers using ARM Instructions

- No breakdown of currently accessible registers.
  - All instructions can access r0-r14 directly.
  - Most instructions also allow use of the PC.

- Specific instructions to allow access to CPSR and SPSR.

- Note : When in a privileged mode, it is also possible to load-store the (banked out) user mode registers to or from memory.

The Program Status Registers (CPSR and SPSRs)



Copies of the ALU status flags (latched if the instruction has the "S" bit set).

* **Condition Code Flags**

N = **N**egative result from ALU flag.
Z = **Z**ero result from ALU flag.
C = ALU operation **C**arried out
V = ALU operation o**V**erflowed

* **Mode Bits**

M[4:0] define the processor mode.

* **Interrupt Disable bits.**

I = 1, disables the IRQ.
F = 1, disables the FIQ.

* **T Bit (Architecture v4T only)**

T = 0, Processor in ARM state
T = 1, Processor in Thumb state

Condition Flags

	Logical Instruction	Arithmetic Instruction
Flag		
Negative (N='1')	No meaning	Bit 31 of the result has been set Indicates a negative number in signed operations
Zero (Z='1')	Result is all zeroes	Result of operation was zero
Carry (C='1')	After Shift operation '1' was left in carry flag	Result was greater than 32 bits
oVerflow (V='1')	No meaning	Result was greater than 31 bits Indicates a possible corruption of the sign bit in signed numbers

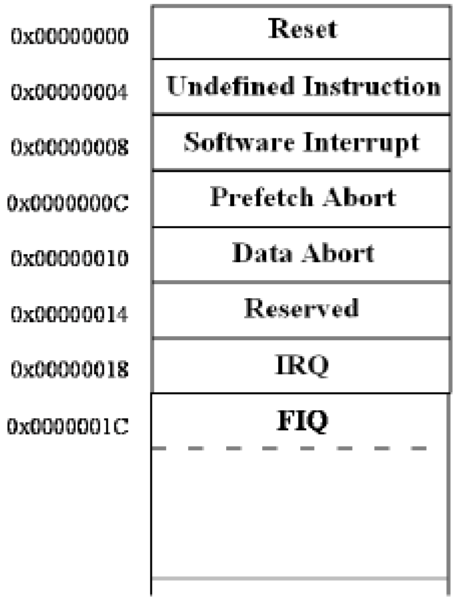
The Program Counter (R15)

- When the processor is executing in ARM state:
  - All instructions are 32 bits in length
  - All instructions must be word aligned
  - Therefore the PC value is stored in bits [31:2] with bits [1:0] equal to zero (as instruction cannot be halfword or byte aligned).
- R14 is used as the subroutine link register (LR) and stores the return address when Branch with Link operations are performed, calculated from the PC.
- Thus to return from a linked branch:

```
MOV r15,r14
   or
MOV pc,lr
```

Exception Handling and the Vector Table

- When an exception occurs, the core:
  - Copies CPSR into SPSR_<mode>
  - Sets appropriate CPSR bits
    - If core implements ARM Architecture 4T and is currently in Thumb state, then
      - ARM state is entered.
    - Mode field bits
    - Interrupt disable flags if appropriate.
  - Maps in appropriate banked registers
  - Stores the “return address” in LR_<mode>
  - Sets PC to vector address
- To return, exception handler needs to:
  - Restore CPSR from SPSR_<mode>
  - Restore PC from LR_<mode>


0x00000000 Reset
0x00000004 Undefined Instruction
0x00000008 Software Interrupt
0x0000000C Prefetch Abort
0x00000010 Data Abort
0x00000014 Reserved
0x00000018 IRQ
0x0000001C FIQ

Conditional Execution

- Most instruction sets only allow branches to be executed conditionally.
- However by reusing the condition evaluation hardware, ARM effectively increases number of instructions.
  - All instructions contain a condition field which determines whether the CPU will execute them.
  - Non-executed instructions consume 1 cycle.
    - Can't collapse the instruction like a NOP. Still have to complete cycle so as to allow fetching and decoding of the following instructions.
- This removes the need for many branches, which stall the pipeline (3 cycles to refill).
  - Allows very dense in-line code, without branches.
  - The Time penalty of not executing several conditional instructions is frequently less than overhead of the branch
    or subroutine call that would otherwise be needed.

The Condition Field

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Instruction Type
Condition				0	0	I	OPCODE				S	Rn				Rs				OPERAND-2												Data processing

0000 = EQ - Z set (equal)

0001 = NE - Z clear (not equal)

0010 = HS / CS - C set (unsigned higher or same)

0011 = LO / CC - C clear (unsigned lower)

0100 = MI -N set (negative)

0101 = PL - N clear (positive or zero)

0110 = VS - V set (overflow)

0111 = VC - V clear (no overflow)

1000 = HI - C set and Z clear (unsigned higher)

1001 = LS - C clear or Z (set unsigned lower or same)

1010 = GE - N set and V set, or N clear and V clear (>or =)

1011 = LT - N set and V clear, or N clear and V set (>)

1100 = GT - Z clear, and either N set and V set, or N clear and V set (>)

1101 = LE - Z set, or N set and V clear,or N clear and V set (<, or =)

1110 = AL - always

1111 = NV - reserved.

Arithmetic Operations

- **Operations are:**

ADD	operand1 + operand2	; Add
ADC	operand1 + operand2 + carry	; Add with carry
SUB	operand1 - operand2	; Subtract
SBC	operand1 - operand2 + carry -1	; Subtract with carry
RSB	operand2 - operand1	; Reverse subtract
RSC	operand2 - operand1 + carry - 1	; Reverse subtract with carry

- **Syntax:**
  - <Operation>{<cond>}{S} Rd, Rn, Operand2
- **Examples**
  - ADD r0, r1, r2
  - SUBGT r3, r3, #1
  - RSBLES r4, r5, #5

Comparisons

- The only effect of the comparisons is to update the condition flags. Thus no need to set S bit.
- Operations are:

CMP	operand1 - operand2	; Compare
CMN	operand1 + operand2	; Compare negative
TST	operand1 AND operand2	; Test
TEQ	operand1 EOR operand2	; Test equivalence

- Syntax:
  - <Operation>{<cond>} Rn, Operand2
- Examples:

```
CMP       r0, r1
TSTEQ     r2, #5
```

Logical Operations

- **Operations are:**

AND	operand1 AND operand2
EOR	operand1 EOR operand2
ORR	operand1 OR operand2
ORN	operand1 NOR operand2
BIC	operand1 AND NOT operand2 [ie bit clear]

- **Syntax:**
  - <Operation>{<cond>}{S} Rd, Rn, Operand2
- **Examples:**

```
AND    r0, r1, r2
BICEQ r2, r3, #7
EORS  r1,r3,r0
```

Data Movement

- Operations are:

  MOV operand2

  MVN NOT operand2

  Note that these make no use of operand1.
- Syntax:
  - <Operation>{<cond>}{S} Rd, Operand2
- Examples:

```
MOV        r0, r1
MOVS       r2, #10
MVNEQ      r1,#0
```

The Barrel Shifter

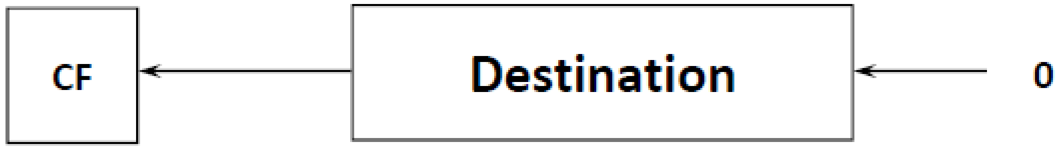
- The ARM doesn’t have actual shift instructions.
- Instead it has a barrel shifter which provides a mechanism to carry out shifts as part of other instructions.
- So what operations does the barrel shifter support?

Barrel Shifter - Left Shift

- Shifts left by the specified amount (multiplies by powers of two) e.g.

  LSL #5 => multiply by 32

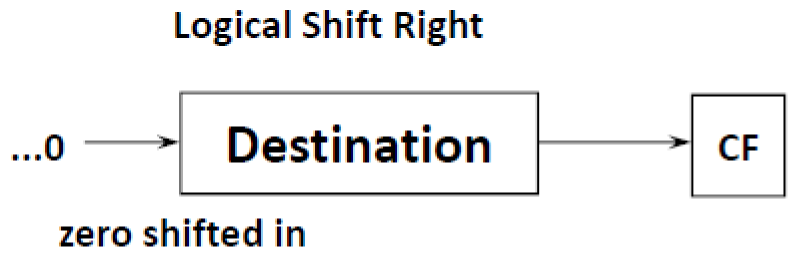
Logical Shift Left (LSL)



Barrel Shifter - Right Shifts

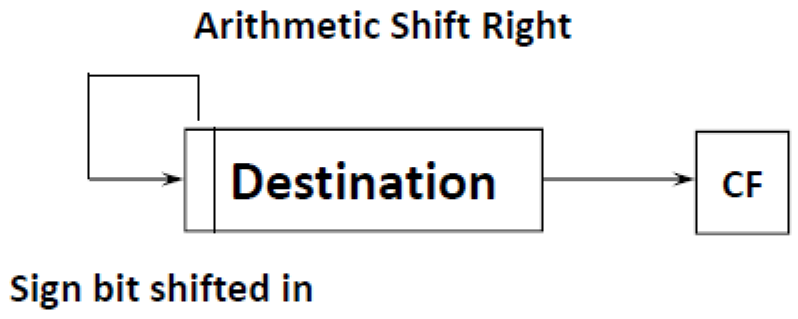
Logical Shift Right (LSR)
Shifts right by the specified amount (divides by powers of two) e.g.

LSR #5 = divide by 32



Arithmetic Shift Right (ASR)
Shifts right (divides by powers of two) and preserves the sign bit, for 2's complement operations. e.g.

ASR #5 = divide by 32



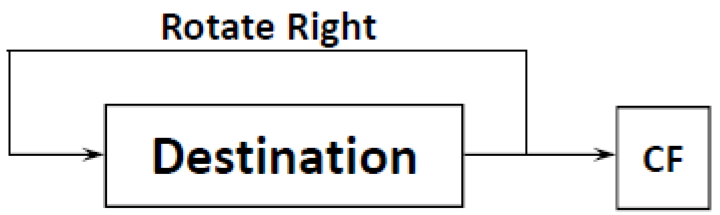
Barrel Shifter - Rotations

Rotate Right (ROR)

Similar to an ASR but the bits wrap around as they leave the LSB and appear as the MSB.

e.g. ROR #5

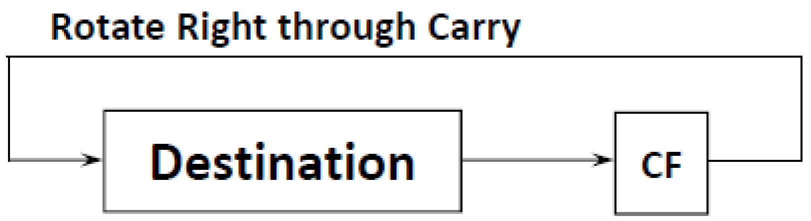
Note the last bit rotated is also used as the Carry Out.



Rotate Right Extended (RRX)

This operation uses the CPSR C flag as a 33rd bit.

Rotates right by 1 bit. Encoded as ROR #0



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 avoided by swapping Rm and Rs around. This works because multiplication is commutative.
  - Cannot use PC.
- These will be picked up by the assembler if overlooked.
- Operands can be considered signed or unsigned
  - Up to user to interpret correctly.

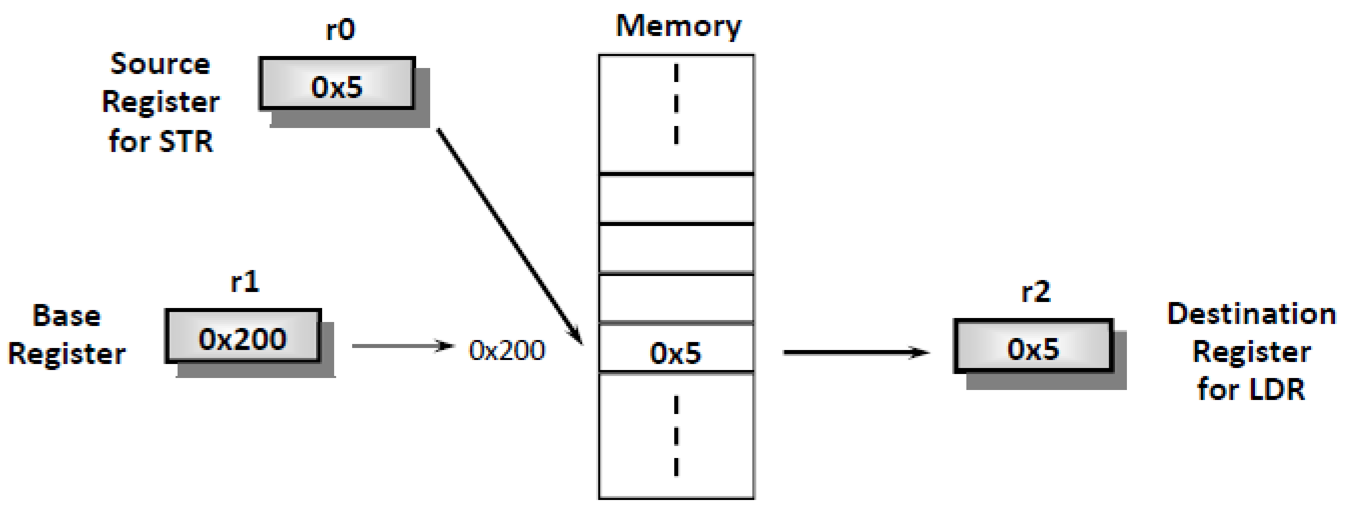
Load / Store Instructions

- 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 it 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).

Load and Store Word or Byte: 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.
```


r0
Source Register for STR
0x5
Memory
r1
Base Register
0x200
0x200
0x5
r2
0x5
Destination Register for LDR