# Microprocessors

Tuba Ayhan

MEF University

## ARM Processor – Instructions 1

Computer Organization and Embedded Systems, Hamacher et. al

# Microprocessors

Tuba Ayhan

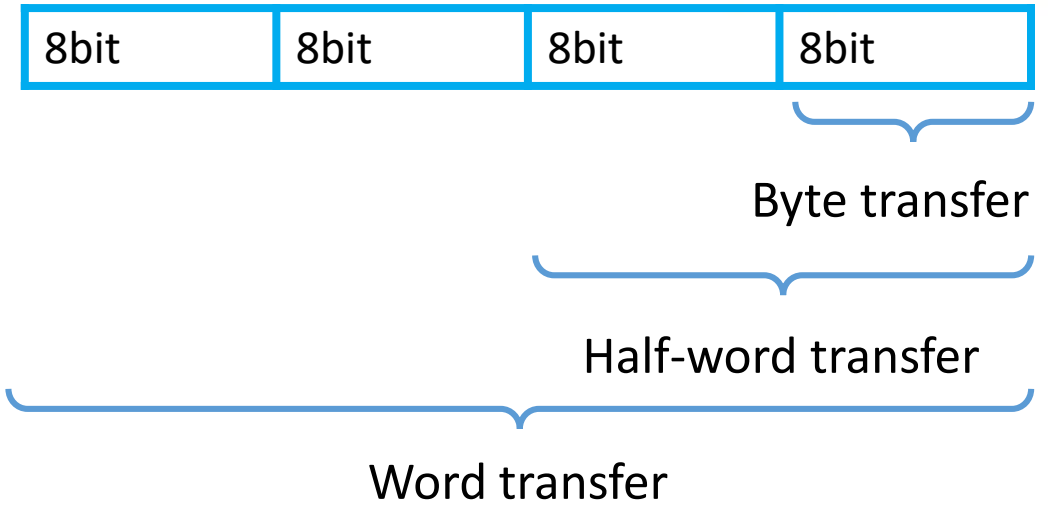
MEF University

## ARM Processor – Instructions

## Load, Store, (Shift, Rotate), Move

Computer Organization and Embedded Systems, Hamacher et. al

# Load and Store Instructions




- Word transfer: LDR, STR
- Byte transfer: low-order byte position of the register.
- Half-word transfer: it is located in the low-order half of the register.
- For Load instructions, byte and half-word values are extended to the 32-bit register lenght
  - zero-extended by LDRB and LDRH
  - sign-extended by LDRSB and LDRSH
- The byte and half-word Store instructions: STRB and STRH.

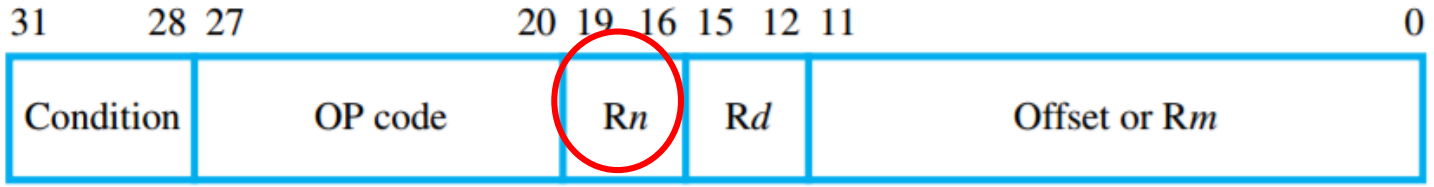
# Loading and Storing Multiple Operands

- Block Transfer instructions:

  LDM (Load Multiple) and STM (Store Multiple)

  LDM *Rsource*, {Rd1, Rd2, Rd3 ….}
- Any subset of the general-purpose registers can be loaded or stored to successive word locations.
- Only **word** operands are allowed: The offset magnitude **is always 4** in these instructions.
- All of the forms of pre- and post-indexing with and without writeback are available.
- The list of registers must be in **increasing order**, but they do not need to be contiguous: LDM R1 , {R4 R5 R9}

# Loading and Storing Multiple Operands




base register

LDM R10!, {R0, R1, R6, R7}

| | |
|---|---|
| 1000 | 48A20123 |
| 1004 | 12300000 |
| 1008 | AAF11001 |
| 1012 | BC4F8A10 |
| | |
| | ..... |
| | |
| | |

| | Before the first transfer | After the last transfer |
|---|---|---|
| R0 | x | 48A20123 |
| R1 | x | 12300000 |
| | | |
| R6 | x | AAF11001 |
| R7 | x | BC4F8A10 |
| | ..... | ..... |
| R10 | 1000 | 1016 |
| | | |

# Load Instructions

- LDM (Load Multiple registers) loads one or more registers from consecutive addresses in memory at an address specified in a base register.
- LDR (Load Register) loads a value from memory to an ARM register, optionally updating the register used to give the address.
- LDRD (Load Register Dual) calculates an address from a base register value and a register offset, loads two words from memory, and writes them to two registers.
- LDREX (Load register exclusive). Performs a load from a location and marks it for exclusive access. Byte, halfword, word and doubleword variants are provided.

# Store Instructions

- STM (Store Multiple registers) writes one or more registers to consecutive addresses in memory to an address specified in a base register.
- STR (Store Register) stores a value to memory from an ARM register, optionally updating the register used to give the address.
- STRD (Store Register Dual) calculates an address from a base register value and a register offset, and stores two words from two registers to memory. It can use offset, post-indexed, or pre-indexed addressing.
- STREX (Store register exclusive). Performs a store to a location marked for exclusive access, returning a status value if the store succeeded. Byte, halfword, word and doubleword variants are provided.

# Refer to Cortex A programming guide.

MEF UNIVERSITY
Your Freedom in Learning

### A.1.40 LDRD

LDRD (Load Register Dual) calculates an address from a base register value and a register offset, loads two words from memory, and writes them to two registers.

#### Syntax

```
LDRD{cond} Rt, Rt2, [{Rn},+/-{Rm}]{!}
LDRD{cond} Rt, Rt2, [{Rn}],+/-{Rm}
```

where:

cond is an optional condition code. See Section 6.1.2.

Rt is the first destination register. This register must be even-numbered and not R14.

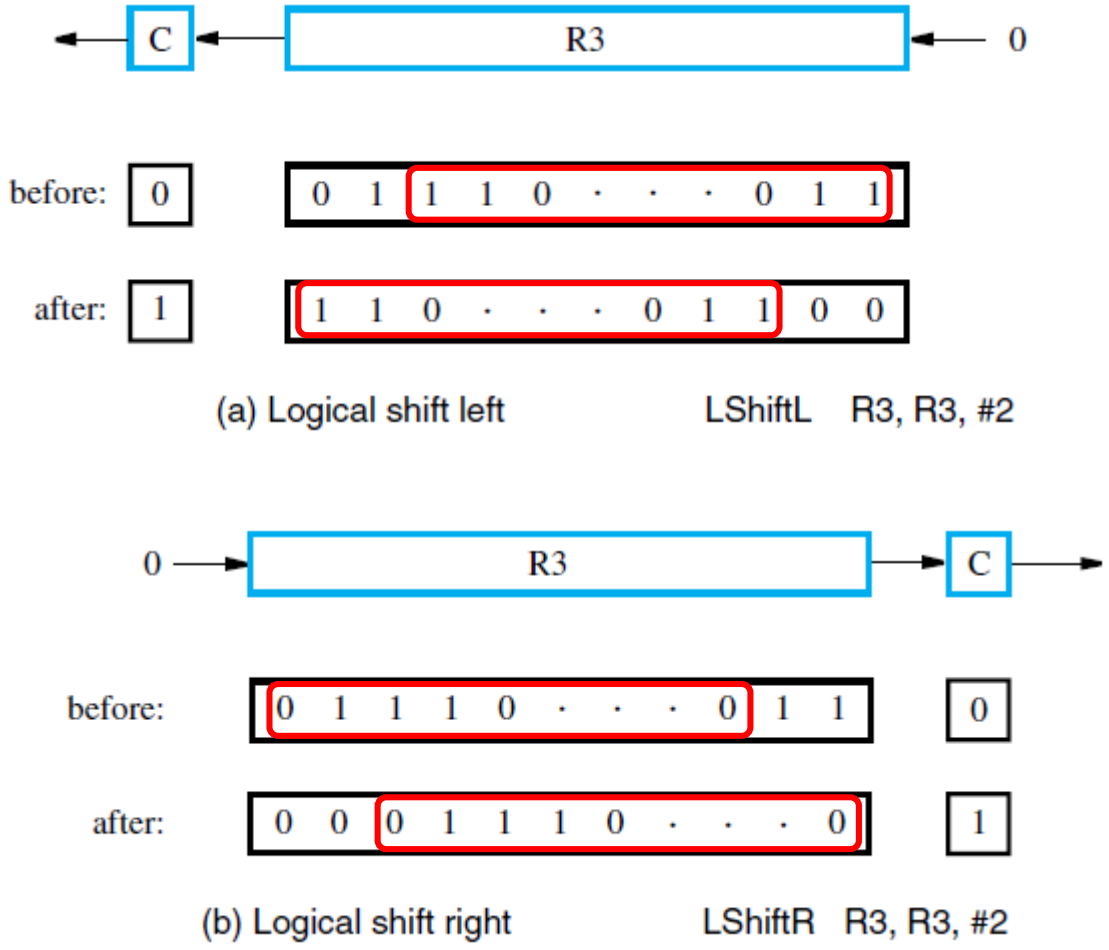
Rt is the second destination register. This register must be <R(t+1)>.

Rn is the base register. The SP or the PC can be used.

+/- is + or omitted if the value of <Rm> is to be added to the base register value (add == TRUE), or – if it is to be subtracted (add == FALSE).

Rm contains the offset that is applied to the value of <Rn> to form the address.

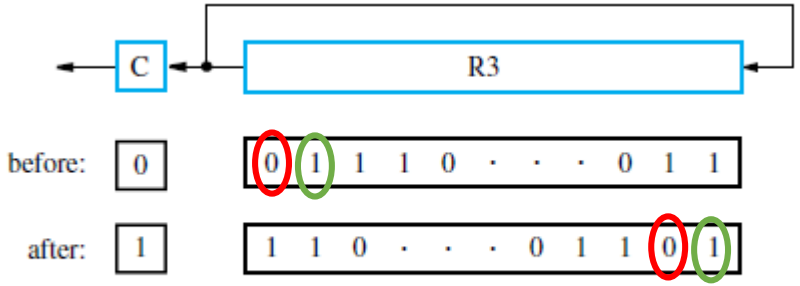
# Logic shift


C
R3
0
before:
0
0 1 1 1 0 · · · 0 1 1
after:
1
1 1 0 · · · 0 1 1 0 0
(a) Logical shift left
LShiftL R3, R3, #2
0
R3
C
before:
0 1 1 1 0 · · · 0 1 1
0
after:
0 0 0 1 1 1 0 · · · 0
1
(b) Logical shift right
LShiftR R3, R3, #2

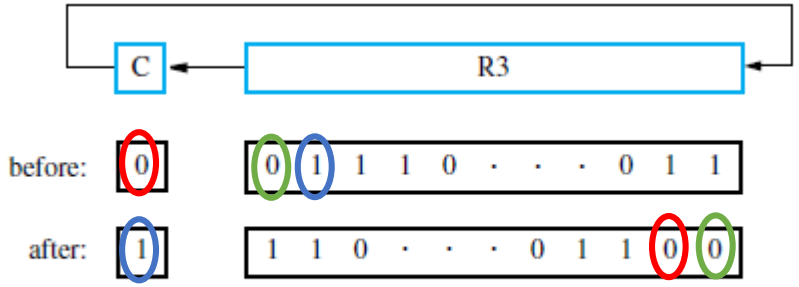
# Arithmetic shift


(c) Arithmetic shift right AShiftR R3, R3, #2

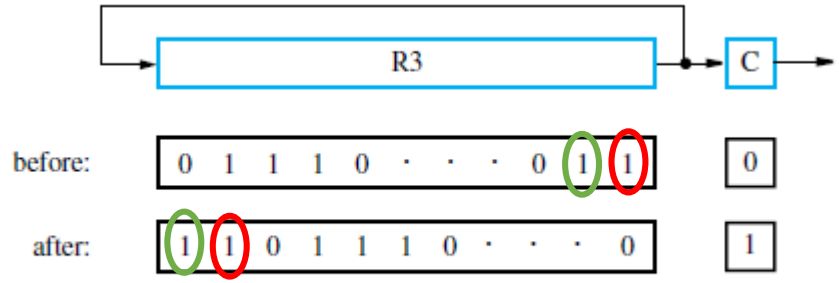
# Rotate



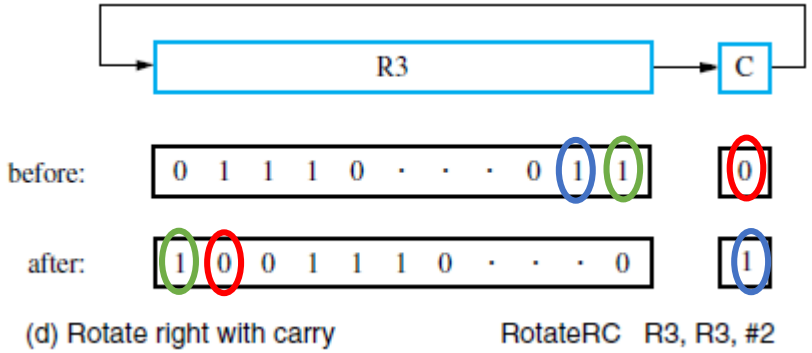

(a) Rotate left without carry RotateL R3, R3, #2




(b) Rotate left with carry RotateLC R3, R3, #2




(c) Rotate right without carry RotateR R3, R3, #2




(d) Rotate right with carry RotateRC R3, R3, #2

# Move Instructions

```
MOV Rd, Rm
```

- copies the contents of register Rm into register Rd.

```
MOV Rd, #value
```

- loads an 8-bit immediate value (an unsigned number in the range 0 to 255) into the destination register.
- *Move Negative:*

```
MVN R0, #4
```

- forms the bit-complement of the source operand before it is placed in the destination register.

# Move Instructions

- Loading negative numbers: **To load –c** into a register, use the MVN instruction with an immediate source operand value of **c – 1**.

```
MVN R0, #4
```

- Loads –5 into register R0.
- MOV R0, #–5 → assembler → MVN R0, #4
- A MOV instruction with a negative immediate source operand is an example of a *pseudoinstruction*.

# Implementing Shift and Rotate Instructions

- ARM processors do not have explicit instructions for shifting or rotating register contents!

```
MOV Ri, Rj, LSL #4
```

- Shift Rj left by 4 bits, and load it into Ri.

# Example – load address

- ARM instructions consist of a single 32-bit word, so the address cannot be represented by an immediate value in a Move instruction.

```
LDR Ri, =ADDRESS   // pseudoinstruction
```

- loads the address value ADDRESS into register Ri.

# Microprocessors

Tuba Ayhan

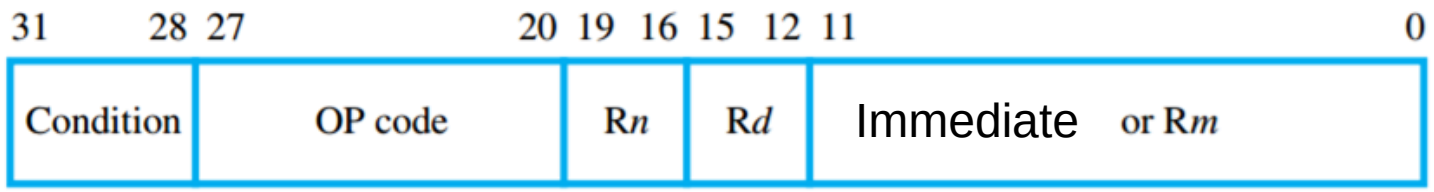
MEF University

## ARM Processor – Instructions

## Arithmetic Instructions

Computer Organization and Embedded Systems, Hamacher et. al

# Arithmetic Instructions – ADD and SUB




- OP Rd, Rn, Rm: OP code is performed on the source operands in general-purpose registers Rn and Rm. The result is placed in destination register Rd.

  ADD R0, R2, R4 R0←[R2] + [R4]

  SUB R0, R6, R5 R0←[R6] − [R5]

  ADD R0, R3, #17 R0←[R3] + 17

- The immediate operand is an 8-bit value. It is an **unsigned** number in the range 0 to 255.

  ADD R0, R3, #−17 →assembler→ SUB R0, R3, #17

# Refer to Cortex A programming guide.

MEF UNIVERSITY
Your Freedom in Learning

### A.1.2 ADD

ADD adds together the values in Rn and Operand2 (or Rn and imm12).

#### Syntax

```
ADD{S}{cond} {Rd,} Rn, <Operand2>

ADD{cond} {Rd,} Rn, #imm12  (Only available in Thumb)
```

where:

S (if specified) means that the condition code flags will be updated depending upon the result of the instruction.

cond is an optional condition code. See Section 6.1.2.

Rd is the destination register.

Rn is the register holding the first operand.

Operand2 is a flexible second operand. See Section 6.2.1.

imm12 is in the range 0-4095.

# Refer to Cortex A programming guide.

MEF UNIVERSITY
Your Freedom in Learning

### A.1.134 SUB

SUB (Subtract) subtracts the value Operand2 from Rn (or subtracts imm12 from Rn).

#### Syntax

```
SUB{S}{cond} {Rd,} Rn, <Operand2>

SUB{cond}{Rd,}, Rn, #imm12 (Only available in Thumb)
```

where:

S (if specified) means that the condition code flags will be updated depending upon the result of the instruction.

cond is an optional condition code. See Section 6.1.2.

Rd is the destination register.

Rn is the register holding the first operand.

Operand2 is a flexible second operand. See Section 6.2.1.

imm12 is in the range 0-4095.

# Shifting or Rotation of the Second Source Operand

- Logical shift left (LSL), logical shift right (LSR), arithmetic shift right (ASR), and rotate right (ROR). The carry bit, C, is not involved in these operations.

```
ADD R0, R1, R5, LSL #4
```

Destination Source1 Source2

R5 is shifted left 4 bit positions : [R5] × 16

# Multiple-Word Operands

- Use ADC (Add with carry) and SBC (Subtract with carry).
- Example: Add two 64 bit numbers:
  - Number 0 = | High-order word0 | Low-order word0 |
  - Number 1 = | High-order word1 | Low-order word1 |

```
ADDS R6, R2, R4        R6 ← [R2] + [R4]
ADC R7, R3, R5         R7 ← [R3] + [R5] +C
```

| | |
|---|---|
| R0 | |
| R1 | |
| R2 | Low-order word0 |
| R3 | High-order word0 |
| R4 | Low-order word1 |
| R5 | High-order word1 |
| | |
| | |
| | |

# Multiplication

- Simple multiplication:

  $\mathrm{MUL\ R0, R1, R2}$ $\quad\quad$ $\mathrm{R0 \leftarrow [R1] \times [R2]}$

- Low-order 32 bits of the product is the result, the high-order bits of the product are discarded.
- Multiply-Accumulate operation:

  $\mathrm{MLA\ R0, R1, R2, R3}$ $\quad$ $\mathrm{R0 \leftarrow ([R1] \times [R2]) + [R3]}$

- Multiply and Multiply-Accumulate instructions for double-length (64-bit) products are also provided.
- There are different versions for signed and unsigned operands.

# Multiplication instructions

- MLS (Multiply and Subtract)
- UMAAL (Unsigned Multiply Accumulate Long; 64 <= 32 + 32 + 32 x 32)
- SMLAxy (Signed Multiply Accumulate; 32 <= 32 + 16 × 16)
- SMLAD (Dual Signed Multiply Accumulate; 32 <= 32 + 16 x 16 + 16 x 16)
- SMLAL (Signed Multiply Accumulate 64 <= 64 + 32 x 32)
- SMLALxy (Signed Multiply Accumulate; 64 <= 64 + 16 x 16)
- SMLALD (Dual Signed Multiply Accumulate Long; 64 <= 64 + 16 x 16 + 16 x 16)
- SMLAW (Signed Multiply with Accumulate Wide; 32 <= 32 x 16 + 32)
- SMLSLD (Dual Signed Multiply Subtract Accumulate Long; 64 <= 64 + 16 x 16 - 16 x 16)
- SMMLA (Signed top word Multiply with Accumulate; 32 <= top word (32 × 32 + 32))
- SMMLS (Signed top word Multiply with Subtract; 32 <= top word (32 × 32 - 32))
- SMMUL (Signed top word Multiply; 32 <= top word (32 × 32 ))
- SMUAD (Dual Signed Multiply and Add products)
- SMUSD (Dual Signed Multiply and Subtract products)
- The SMULxy (Signed Multiply (32 <= 16 × 16)
- The SMULL (signed multiply long; 64 <= 32 x 32)
- SMUAD (Dual Signed Multiply and Add products)
- SMUSD (Dual Signed Multiply and Subtract products)