# Microprocessor & Computer Architecture (µpCA)

UE19CS252

**Dr. D. C. Kiran**

Department of
Computer Science and Engineering

# Microprocessor & Computer Architecture (µpCA)

## Barrel Shifter

**Dr. D. C. Kiran**

Department of Computer Science and  Engineering

# Microprocessor & Computer Architecture (μpCA)

## Syllabus

**Unit 1: Basic Processor Architecture and Design**

- ~~Microprocessor Overview~~
- ~~CISC VS RISC~~
- ~~Introduction to ARM Processor & Applications~~
- ~~ARM Architecture Overview~~
- ~~Different ARM processor Modes~~
- ~~Register Bank~~
- ~~ARM Program structure~~
- ~~ARM Instruction Format~~
- **ARM INSTRUCTION SET**

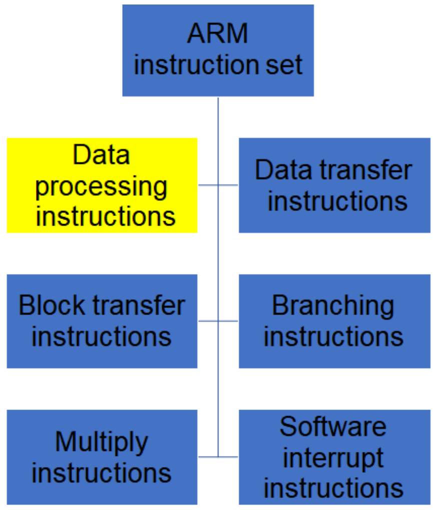
  Data Processing Instructions

  **~~Data Movement Instruction~~**

  **~~Arithmetic Instruction~~**

  **~~Multiword Arithmetic~~**

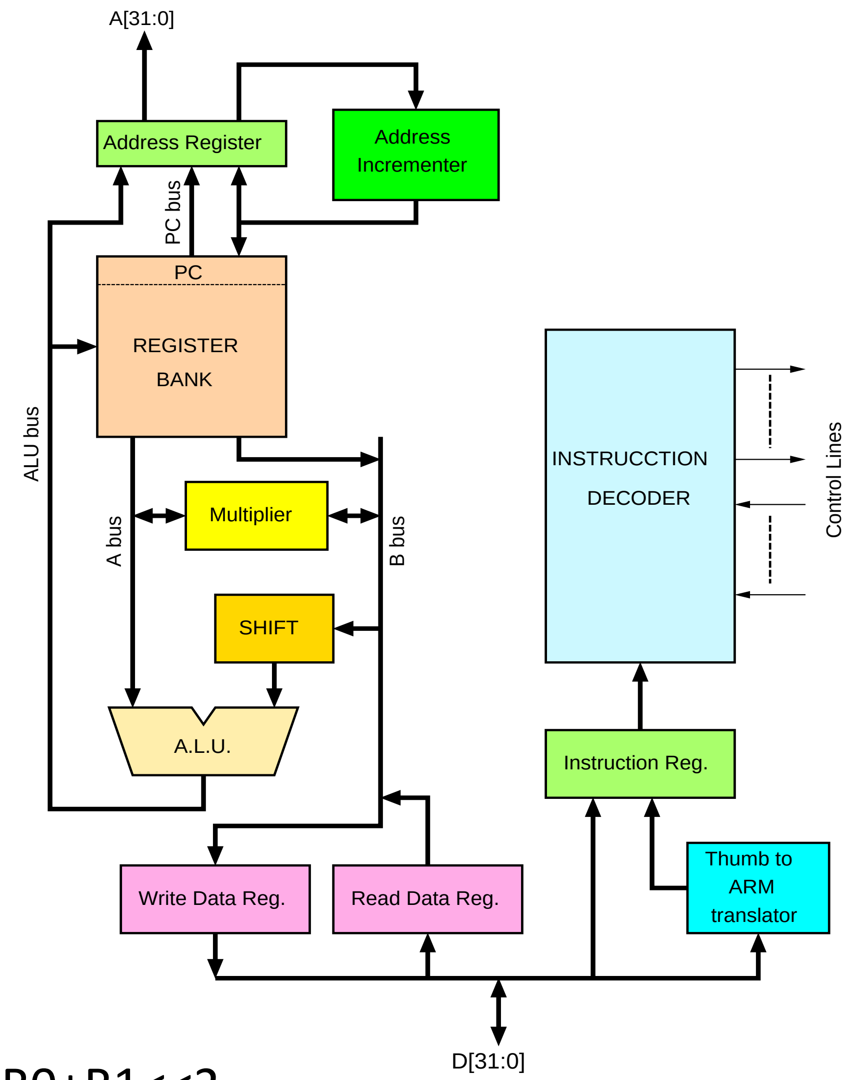
  **Barrel Shifter**


ARM instruction set
Data processing instructions
Data transfer instructions
Block transfer instructions
Branching instructions
Multiply instructions
Software interrupt instructions

# Microprocessor & Computer Architecture (μpCA)

## Barrel Shifter

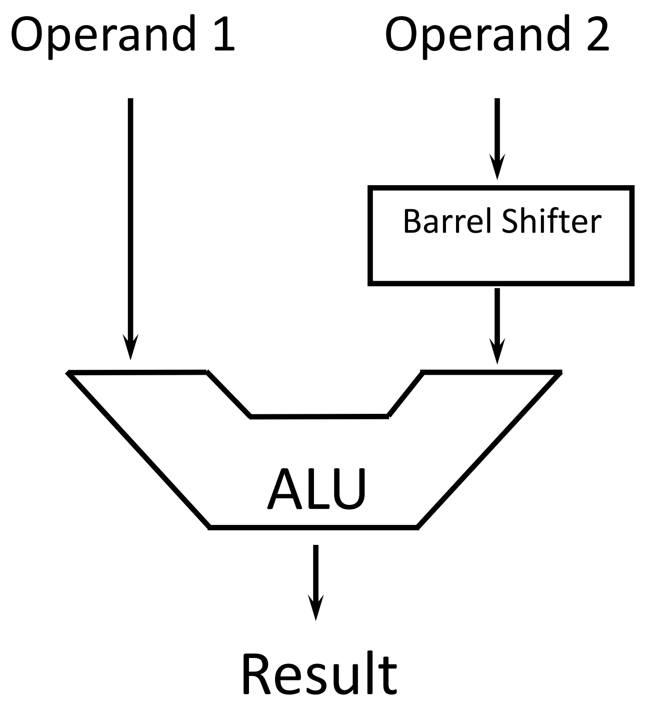
- The ARM doesn’t have actual shift instructions.
- Instead, it has a barrel shifter which provides a mechanism to carry out shifts as part of other instructions.



- Example:

ADD R3,R0,R1, LSL#2 // R3=R0+R1<<2

## Barrel Shifter

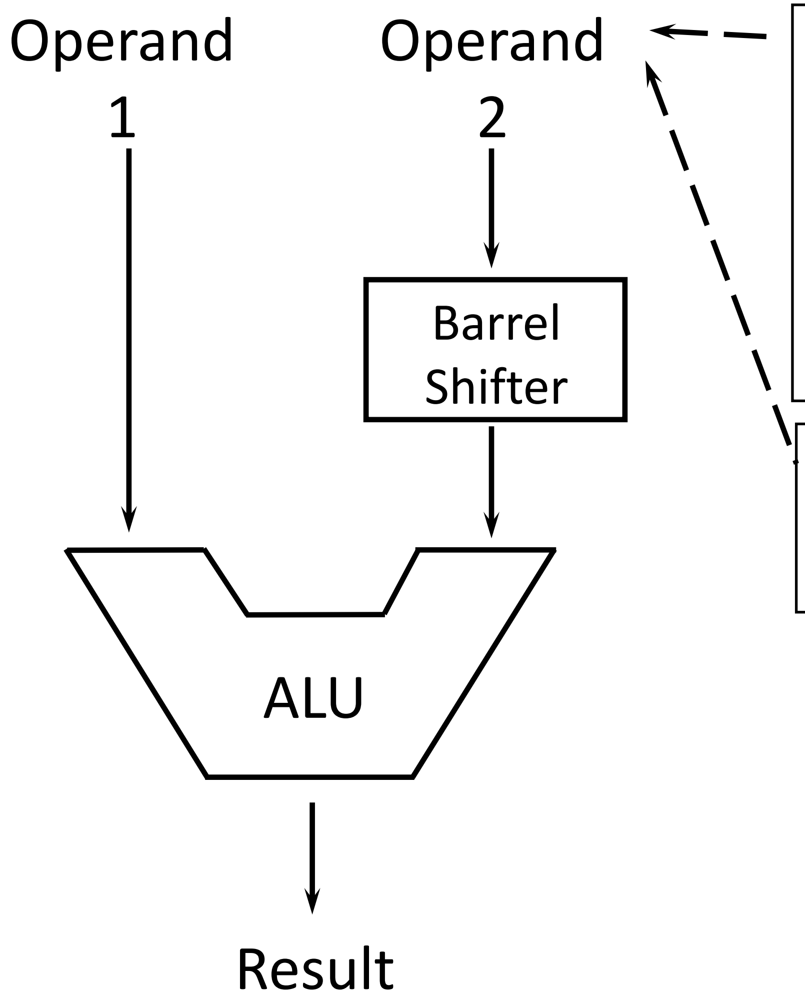


**Possible Shift Operations:**
**LSL:** Left Shift
**LSR:** Right Shift
**ASR:** Arithmetic Right Shift
**ROR:** Rotate Right
**RRX:** Rotate Right Extended

- If no shift is specified then a default shift is applied: LSL #0
    - i.e. barrel shifter has no effect on value in register.

# Microprocessor & Computer Architecture (μpCA)

## Barrel Shifter



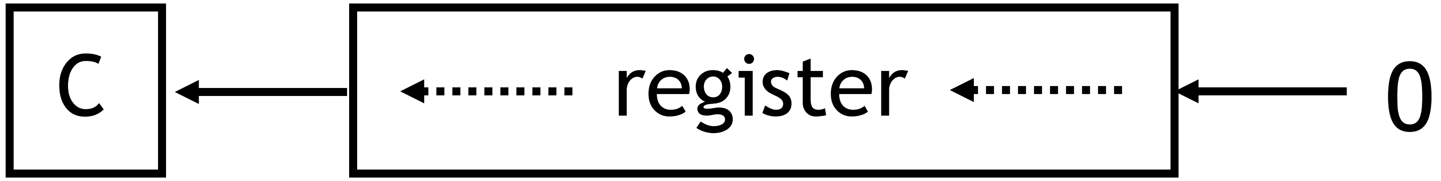
- Register, optionally with shift operation applied.
- Shift value can be either be:
  - 5 bit unsigned integer
  - Specified in bottom byte of another register (7th to 11th bit)

* Immediate value

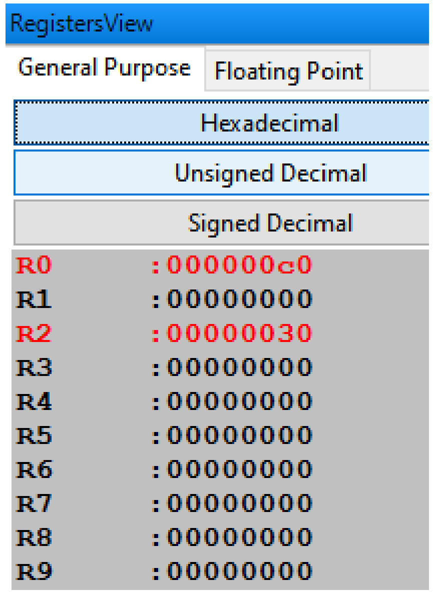
- 8 bit number

## Microprocessor & Computer Architecture (μpCA)

### Logical shift left



```
MOV  R0, R2, LSL #2 @ R0:=R2<<2
                    @ R2 unchanged
```


RegistersView
General Purpose | Floating Point
Hexadecimal
Unsigned Decimal
Signed Decimal

```
R0      :000000c0
R1      :00000000
R2      :00000030
R3      :00000000
R4      :00000000
R5      :00000000
R6      :00000000
R7      :00000000
R8      :00000000
R9      :00000000
```

**Example:**

```
00000000 00000000 00000000 00110000
Before R2=0x00000030
After R2=0x00000030
      R0=0x000000C0
00000000 00000000 00000000 11000000
```

# Microprocessor & Computer Architecture (μpCA)

## Barrel Shifter – Logical Left Shift (LSL)

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

LSL #5 = multiply by 32

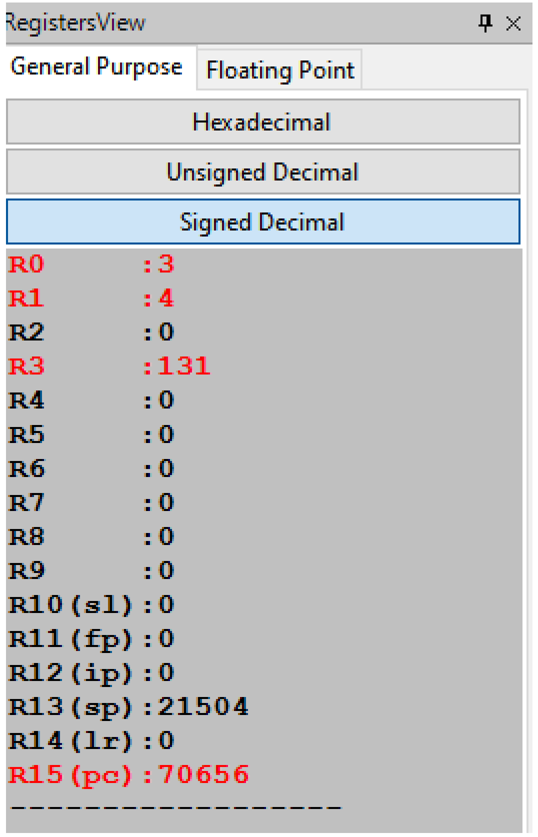
```
.text
MOV R0, #3
MOV R1, #4
ADD R3,R0,R1,LSL#5
.end
```

R1 Before= 00000100

R1 After =10000000= 128

**Output**

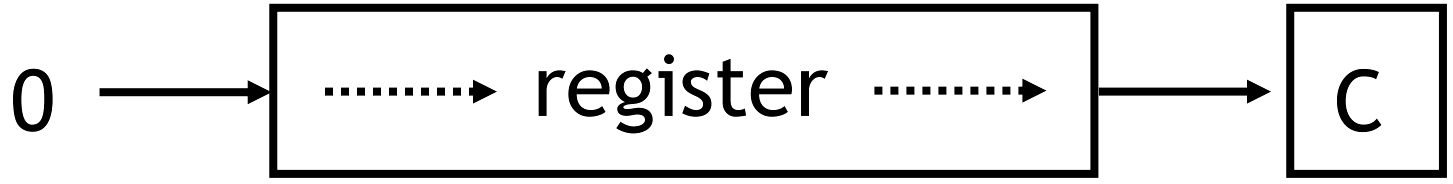
R3 will have 131



```
R0      :3
R1      :4
R2      :0
R3      :131
R4      :0
R5      :0
R6      :0
R7      :0
R8      :0
R9      :0
R10(sl):0
R11(fp):0
R12(ip):0
R13(sp):21504
R14(lr):0
R15(pc):70656
-------------------
```

## Microprocessor & Computer Architecture (μpCA)

### Logical shift right (LSR)



```
MOV  R1, R2, LSR #2 @ R1:=R2>>2
                    @ R2 unchanged
```

**Example:**

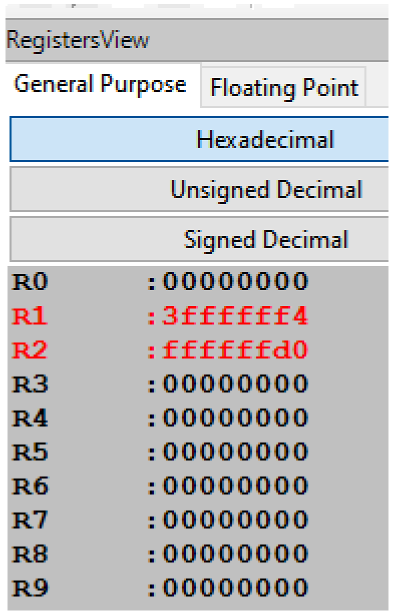
11111111 11111111 11111111 11010000

Before R2=0xffffffd0

After  R1=0x3ffffff4

00111111 11111111 11111111 11110100

R2=0xffffffd0



```
R0   :00000000
R1   :3ffffff4
R2   :ffffffd0
R3   :00000000
R4   :00000000
R5   :00000000
R6   :00000000
R7   :00000000
R8   :00000000
R9   :00000000
```

## Barrel Shifter - Right Shifts

```
.text
MOV R0, #3
MOV R1, #256
ADD R3,R0,R1,LSR #5
.end
```

**Output**

R3 will have 11

**LSR is division 2^n**
**256/32**

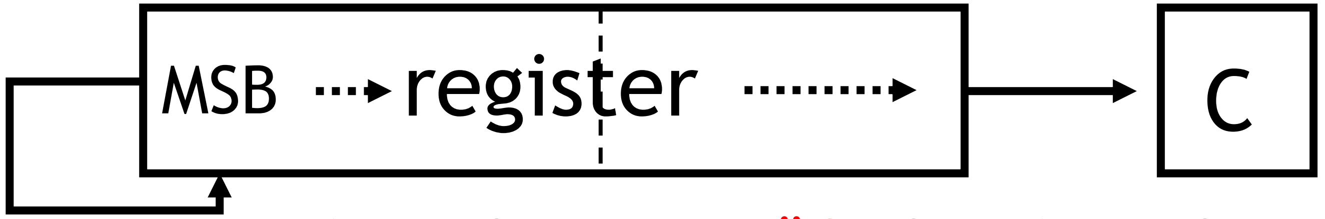
**Before R1: 00000000 00000000 000000 1000 0001**

**After Right Shift R1:00000000 00000000 000000 0000 1000 (8)**
**R0:00000000 00000000 000000 0000 0011 (3)**

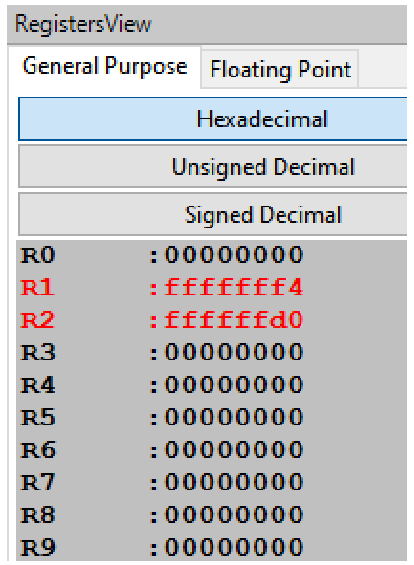
**R3: 00000000 00000000 000000 0000 1011 (11)**

## Microprocessor & Computer Architecture (μpCA)

### Arithmetic shift right (ASR)



```
MOV   R1, R2, ASR #2 @ R1:=R2>>2
                     @ R2 unchanged
```

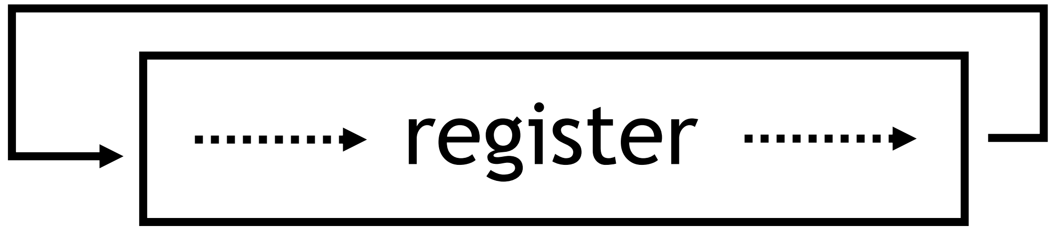


```
R0          :00000000
R1          :fffffff4
R2          :ffffffd0
R3          :00000000
R4          :00000000
R5          :00000000
R6          :00000000
R7          :00000000
R8          :00000000
R9          :00000000
```

```
Example:
11111111 11111111 11111111 11010000
Before R2=0xffffffd0
After  R1=0xfffffff4
11111111 11111111 11111111 11110100
       R2=0xffffffd0
```

## Microprocessor & Computer Architecture (μpCA)

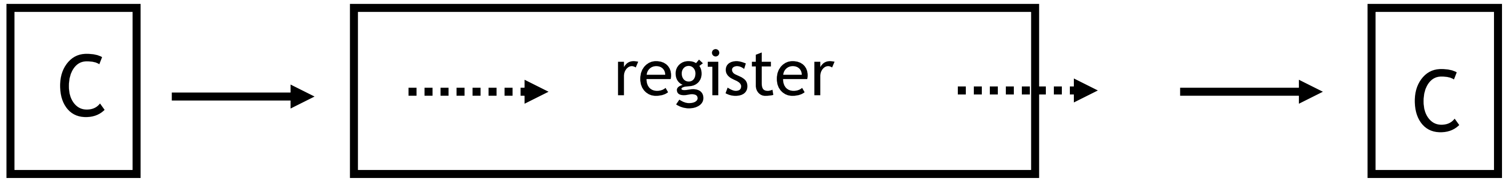
### Rotate right (ROR)



```
MOV  R0, R2, ROR #2 @ R0:=R2 rotate
                    @ R2 unchanged
Example:
11111111 11111111 11111111 11010101
Before R2=0xffffffd5
After  R0=0x7ffffff5
(01111111 11111111 11111111 11110101)
       R2=0xffffffd4
```

## Microprocessor & Computer Architecture (μpCA)

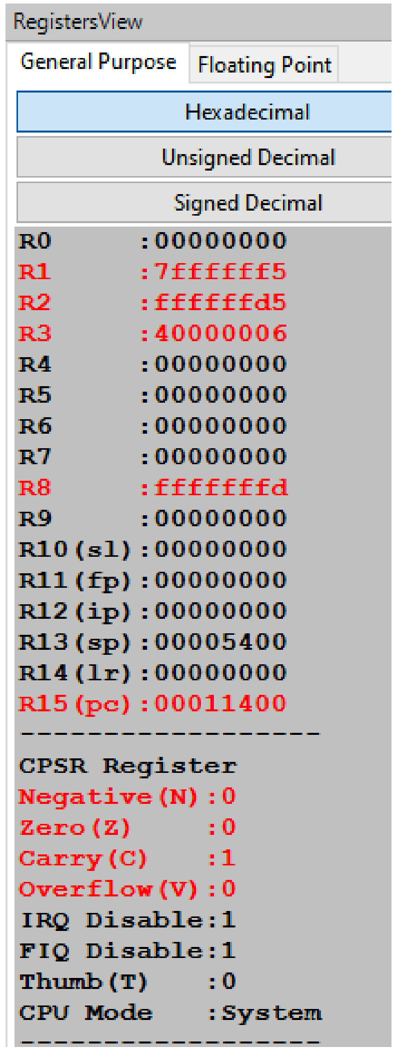
### Rotate right extended (RRX)



```
MOV  R0, R2, RRX     @ R0:=R2 rotate
                     @ R2 unchanged
Example:
11111111 11111111 11111111 11010101
Before R2=0xffffffd1, C=1
After  R0=0x7ffffff7, C=1
01111111 11111111 11111111 11110101
       R2=0xffffffd1
```

## Microprocessor & Computer Architecture (μpCA)

```
.text
mov r2,#0xffffffd5
adds r3,r2,#0x40000031
mov r1,r2,ror #2
.end
```



```
R0      :00000000
R1      :7ffffff5
R2      :ffffffd5
R3      :40000006
R4      :00000000
R5      :00000000
R6      :00000000
R7      :00000000
R8      :fffffffd
R9      :00000000
R10(sl):00000000
R11(fp):00000000
R12(ip):00000000
R13(sp):00005400
R14(lr):00000000
R15(pc):00011400
------------------
CPSR Register
Negative(N):0
Zero(Z)    :0
Carry(C)   :1
Overflow(V):0
IRQ Disable:1
FIQ Disable:1
Thumb(T)   :0
CPU Mode   :System
------------------
```

## THINK ABOUT IT : MULTIPLICATION BY CONSTANT USING THE BARREL SHIFTER

- **Multiplication by 2^n (1,2,4,8,16,32..)**

```
MOV Ra, Rb, LSL #n
```

- **Multiplication by 2^n+1 (3,5,9,17..)**

```
ADD Ra,Ra,Ra,LSL #n
```

- **Multiplication by 2^n-1 (3,7,15..)**

```
RSB Ra,Ra,Ra,LSL #n
```

- **Multiplication by 6**

```
ADD Ra,Ra,Ra,LSL #1     ; multiply by 3
MOV Ra,Ra,LSL#1         ; and then by 2
```

- **Multiply by 10 and add in extra number**

```
ADD Ra,Ra,Ra,LSL#2 ; multiply by 5
ADD Ra,Rc,Ra,LSL#1 ; multiply by 2 and add in next digit
```

## Summary : Arithmetic Operations Addressing Mode

1. Register Direct Addressing: Operand values are in registers:

   ADD r3, r0, r1; r3=r0+r1

2. Immediate Addressing Mode: Operand value is within the instruction

   ADD r3, r0, #7; r3=r0+7

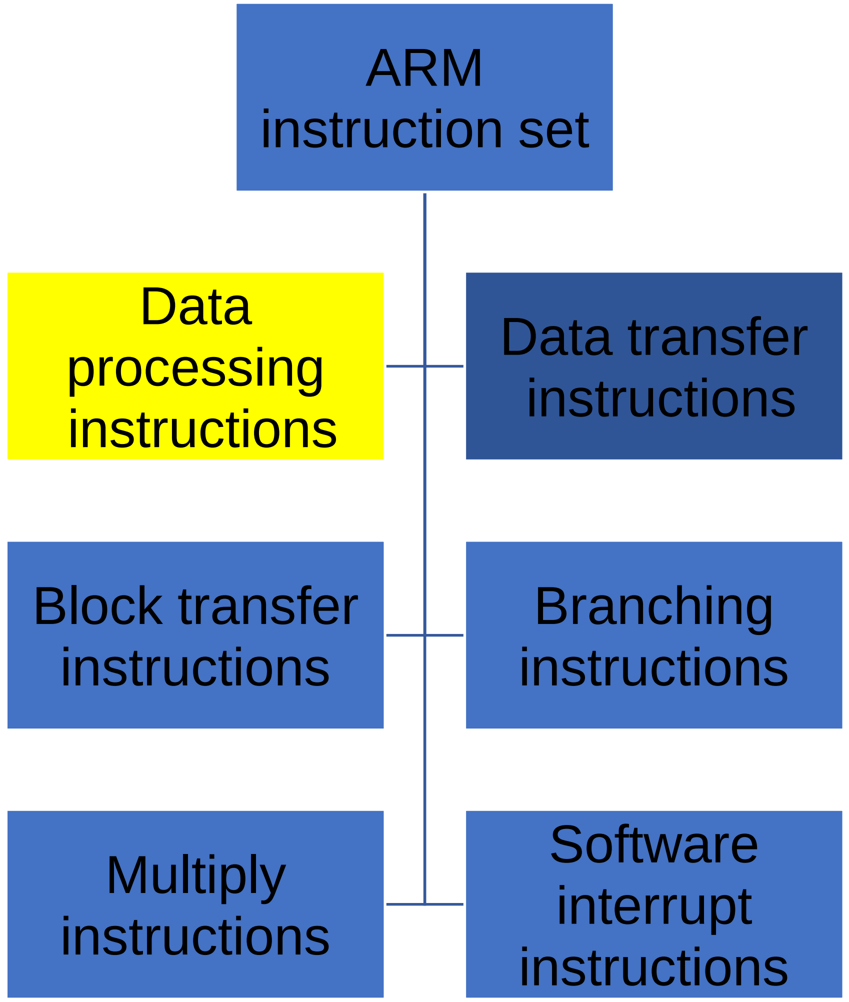
   The number 7 is stored as part of the instruction

3. Register direct with shift or rotate (more next lecture)

   ADD r3, r0, r1, LSL#2; r3=r0+ r1<<2

## Microprocessor & Computer Architecture (μpCA)

### Next Class: Logical and Comparison Instructions


ARM instruction set
Data processing instructions
Data transfer instructions
Block transfer instructions
Branching instructions
Multiply instructions
Software interrupt instructions

## THANK YOU

**Dr. D. C. Kiran**
Department of Computer Science and Engineering

**dckiran@pes.edu**

9829935135