Module - 2

1. Explain in detail about data processing instruction in ARM.

Ans

→ Move instruction :-

* Simplest ARM instruction
* Copies N into a destination register Rd, where N is a register or immediate value

Syntax :-

<instruction> {<cond>} {S} Rd, N

eg :- Pre :- r5 = 5
r7 = 8
MOV r7, r5

POST r5 = 5
r7 = 5

→ Arithematic instruction :-

* implements addition subtraction of 32-bit Signed & unsigned value.

Syntax :- <instruction> {<cond>} {S} Rd, Rn, N

eg :- PRE r0 = 0x0000 0000
r1 = 0x0000 0002
r2 = 0x0000 0001
SUB r0, r1, r2

POST r0 = 0x0000 0001

→ Logical instruction

* Performs bitwise logical operations on the 2 source register.

Syntax:- <instruction>{<cond>}{S} Rd, Rn, N

eg:- r0 = 0x0000 0000

r1 = 0x02040608

r2 = 0x10305070

ORR r0, r1, r2

POST r0 = 0x12345678

→ Comparision Instruction

* Used to compare a register with a 32-bit value
* Update the cpsr flag bits acc. to the result but do not affect other register.

Syntax:- <instruction>{<cond>} Rn, N

eg:- PRE cpsr = nzcpqiFt-USER

r0 = 4

r9 = 4

CMP r0, r9

POST cpsr = nzcpqiFt-USER.

→ Multiply instructions.

* Multiply ~~instruction~~ the contents of a pair of registers and depending upon the instruction, accumulate the register in with another register.
* The final result is placed in destination register.

Syntax:-

MUL {<cond>} {S} Rd, Rm, Rs ⇒ Rd = Rm * Rs

MLA {<cond>} {S} Rd, Rm, Rs, Rn ⇒ Rd = (Rm * Rs) + Rn

eg:- PRE r0 = 0x0000 0000
r1 = 0x0000 0002
r2 = 0x0000 0002

MUL r0, r1, r2

POST r0 = 0x0000 0004
r1 = 0x0000 0002
r2 = 0x0000 0002

2) List & Explain all logical & Arithematic shift & rotate instruction.

→ Logical Shift Left (LSL):- This operation shift the bits of register Rm to the left by a specified no. of bits indicated by <Shift>.

* Zeros are shifted in from the right & bits shifted out from the left are discarded.

Syntax:- LSL Rd, Rm, #<Shift>

eg:- Rm = 10101010
#3
Rm = 01010000

→ Logical Shift Right (LSR) :- This operation shifts the bits of the register Rm to the right by a specified no. of bits indicated by <Shift>

* Zeros shifted in from left & bits shifted out from the right are discarded.

Syntax :- LSR Rd, Rm, #<Shift>

eg :- Rm = 00110011 or 11001100
#2 ⇓
00110011

Rd = 00001100

→ Arithematic Shift right (ASR) :- This operation shifts the bits of Register Rm to the right by a specified no. of bits indicated by <Shift>. while preserving the sign bit.

* Sign bit is replicated to maintain the sign of the no.

Syntax :- ASR Rd, Rm, #<Shift>

eg :- Rm = 11100100
#2

Rd = 11111001

→ Rotate Right (ROR) :- This operation rotates the bits of the register Rm to the right by a specified no. of bits indicated by <Shift>

* The bits that are shifted out from the right are rotated back to the left & inserted into the leftmost positions.

Syntax :- ROR Rd, Rm, #<Shift>

Rm = 10101100

#2

Rd = 00101011

→ Rotate Right with Extend (RRX)

* This operation rotates the bits of the register Rm to the right by one bit position & the Carry flag (C) is extended into the vacated most significant bit (MSB) position.

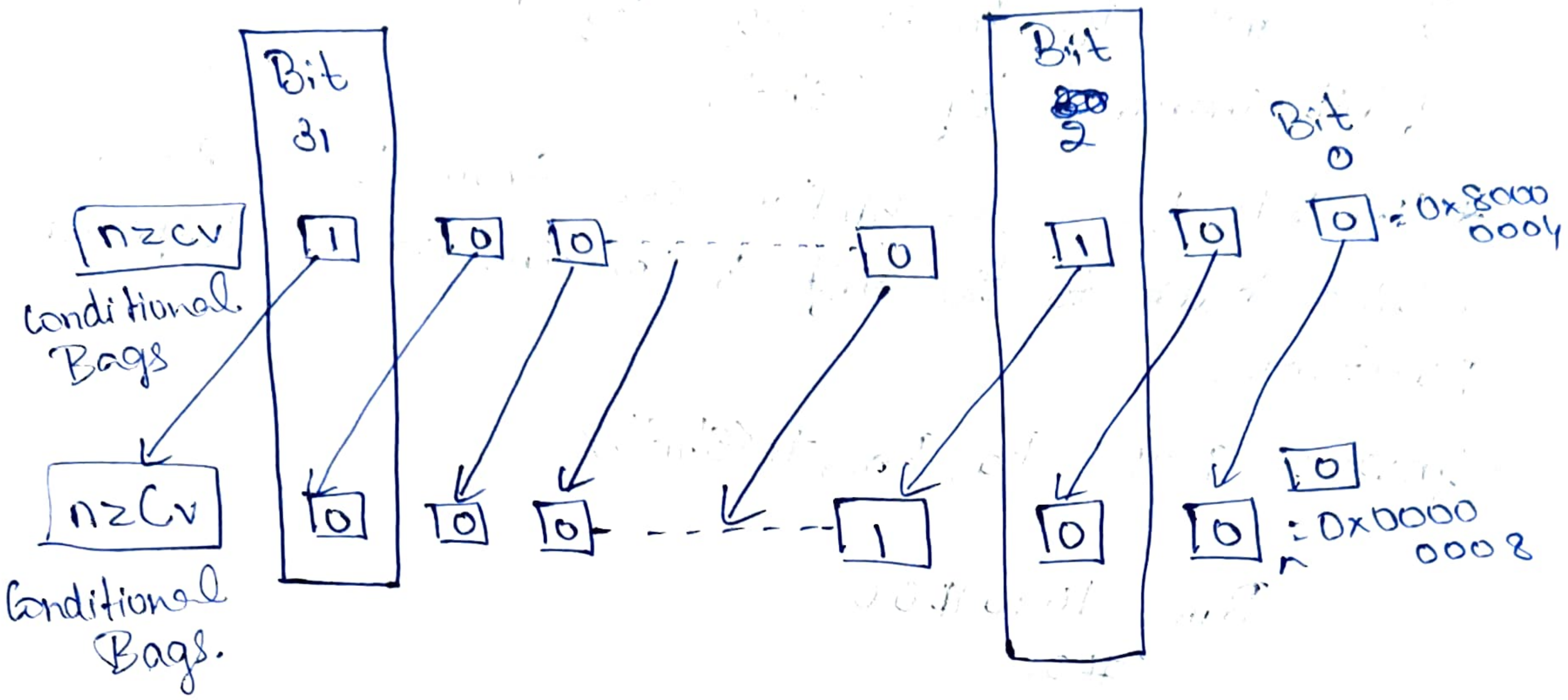
Syntax :- RRX Rd, Rm

eg :- Rm = 10011011

Carry flag = 1

11001101

3) Diagram for logical Shift Left.



→ Contents of Bit 0 are Shifted to Bit 1.

→ Bit 0 is cleared.

→ The C flag is updated with last bit Shifted out of the register.

4) ~~Define Barrel shifter? which are diff. barrel shifter operations.~~

5) Describe various logical instruction?

→ Logical AND Operation.

* Syntax :- AND {S} {cond} Rd, Rn, Operand2

eg:- AND R1, R2, #0xFF

* R1 is destination register, R2 is first operand & 0xFF is immediate value.

* Say R_2 Contains 11001100

* Result is 11001100 AND 1111 1111 = 11001100

→ Logical ORR operation

* Syntax :- ORR {S} {cond} Rd, Rn, operand2

Eg :- ORR R3, R1, #0x00FF

* R3 is destination register, R1 is first operand #0x00FF is immediate value representing 255 in Hexadecimal.

* It performs bitwise OR operation blw the value in register R1 & immediate value #0x00FF and store result in R3.

eg :- Let value of R1 is #0xAAAA(1010101010101010) after executing this instruction R3 would Contain 0xAAFF i.e. (1010101011111111)

→ Logical EOR instruction.

* Syntax :- EOR {S} {cond} Rd, Rn, Operand2

eg :- EOR R3, R1, R2

* It performs a bitwise ~~operation~~ Exclusive OR operation blw R1 & R2 & stores in R3.

* Exclusive OR operation compares each pair of corresponding bits in the operand.

* If bits are diff. result is 1 or else 0

eg:- R1 = 01010101

R2 = 11001100

```
01010101
EOR
11001100
--------
10011001
```

→ BIC [Bit clear] logical operation instruction

* It performs a bitwise logical AND operation between the contents of 2 registers & then clears the corresponding bit in destination register.

Syntax :- BIC {S} {cond} Rd, Rn, Operand2

eg:- BIC R3, R1, R2

R1 = 11001100

R2 = 10101010

R3 = 10001000

8) Explain four Branch instructions

→ B {<cond>} label

→ BL {<cond>} label

→ BX {<cond>} Rm

→ BLX {<cond>} label / Rm

→ B = Branch pc = label

BL = Branch with link pc = label
lr = add. of next instruction after BL

Bx = Branch exchange pc = Rm & 0xfffffffe, T = Rm & 1

BLX = Branch exchange with link pc = Label, T = 1
pc = Rm & 0xfffffffe, T = Rm & 1
lr = add. of next instruction after BLX

The Bx instruction uses an absolute address stored in Register Rm
- ↳ Primarily used to branch to & from Thumb code.
- ↳ T bit of cpsr is updated by least significant bit of the branch register.

The BLX instruction updates the T bits by least significant bit & and additionally. Sets the link register with return address.

7) Explain any 4 arithematic instruction.

→ ADD

* Syntax:- ADD {destination}, {opd1}, {opd2}

eg:- Add R1, R2, R3; Adds the contents of register R2 and R3 & stores the result in R1

→ ADC [Add with Carry]

* Syntax:- ADC {s} Rd, Rn, Opd2

* Here {s} is an optional suffix to update conditional flags, Rd is destination register, Rn is first operand register & Opd2 can be either register or an value.

eg:- ADC R1, R2, R3

Here R2 = 10 (0x0A)

R3 = 15 (0x0F)

carray flag = 1

∴ R1 = R1 + R2 + CF

= 10 + 15 + 1

= 26 (0x1A)

→ SUB: Used to subtract one operand from another.

* Syntax:- SUB {s} {Rd}, {Rn}, opd2

* SUB R1, R2, R3 ; Subtract the value in R3 from the value in R2 & store the result in R1

This instruction subtracts the value in R3 from the value in R2 & stores the result in R1

→ SBC [Subtract with Carry]

* Syntax :- SBC{S} Rd, Rn, opd2

* SBCS R2, R3, R4; Subtract the value in R4 from R3 along with the carry flag and store the result in R2. Update condition flag.

* The value of R4 is subtracted from the value in R3 including the carry flag & the result is stored in R2

eg:- R0 = 10
R1 = 5
SUBS R2, R0, R1 [10-5=5 . Set flag]

11) Explain Load store instruction

→ Transfer data blw memory & processor registers.

→ These are three types of load - store instructions

* Single register transfer.
* Multiple register transfer.
* Swap.

→ Single - Register Transfer

* Used for moving a single data item in & out of register
* The datatypes supported are Signed & unsigned words, halfwords & bytes

Syntax :- <LDR | STR>{<cond>}{B} Rd, addressing1
LDR{<cond>}SB | H | SH Rd, addressing2
STR{<cond>}H Rd, addressing2

→ Multiple Register Transfer.

* Transfer multiple register b/w memory & the processor in single instruction
* The transfer occurs from a base address register Rn pointing into memory.
* More efficient from single-register transfer for moving blocks of data around memory & saving & restoring context and stacks.

Syntax:-

<LDM/STM>{<cond>} <addressing mode> Rn{!}, <registers>{^}

Explain following Instruction:-

→ MLA:- This instruction that performs a multiplication of 2 operands, adds the result to a third operand & Stores the final result in destination register

Syntax:- MLA{S}{cond} Rd, Rn, Rm, Ra

* {S} is optional suffix specifying whether the instruction updates the conditional flags.
* {Cond} is an optional conditional code.
* Rd is destination register
* Rn & Rm are containing values to be multiplied
* Ra is register containing value to be added to Result of Rn & Rm

eg:- MLA R3, R1, R2, R4

Suppose R1 = 2 R2 = 2 R4 = 3

R3 = {2 * 2} + 3

= 4 + 3 = 7

→ MUL :- MUL instruction is used to perform unsigned integer multiplication

* It multiplies 2 registers operands together & store the result in another register.

Syntax :- MUL {S} {cond} Rd, Rn, Rm

* Rd = destination register.
* Rn & Rm = Registers that contain values.

Examples for.

;Y MLA

```
        ENTRY
START   MOV R0, #6
        MOV R1, #5
        MOV R2, #1
        MLA R3, R0, R1, R2
BACK    B BACK
        END
```

→ Multiple Register Transfer.

* Transfer multiple register b/w memory & the processor in single instruction
* The transfer occurs from a base address register Rn pointing into memory.
* More efficient from single-register transfer for moving blocks of data around memory & saving & restoring context and stacks.

Syntax:-

<LDM/STM>{<cond>} <addressing mode> Rn{!}, <registers>{^}

Explain following Instruction:-

→ MLA:- This instruction that performs a multiplication of 2 operands, adds the result to a third operand & Stores the final result in destination register

Syntax:- MLA{S}{cond} Rd, Rn, Rm, Ra

* {S} is optional suffix specifying whether the instruction updates the conditional flags.
* {Cond} is an optional conditional code.
* Rd is destination register
* Rn & Rm are containing values to be multiplied
* Ra is register containing value to be added to Result of Rn & Rm

eg:- MLA R3, R1, R2, R4

Suppose R1 = 2 R2 = 2 R4 = 3

R3 = {2 * 2} + 3

= 4 + 3 = 7

→ MUL :- MUL instruction is used to perform unsigned integer multiplication

* It multiplies 2 register operands together & store the result in another register.

Syntax :- MUL {S} {cond} Rd, Rn, Rm

* Rd = destination register.
* Rn & Rm = Registers that contain values.

Examples for.

i) MLA

```
ENTRY
START   MOV R0, #6
        MOV R1, #5
        MOV R2, #1
        MLA R3, R0, R1, R2
BACK    B BACK
        END
```

;;y MUL

```
ENTRY
START
    MOV R0,#5
    MOV R1,#2
    MUL R2,R0,R2
BACK B BACK
END
```

⇒ TEQ [Test Equivalence]

* It performs as bitwise Exclusive OR[XOR] operation b/w a register value & an operand & updates the conditional flag based on result

* Syntax :- TEQ{cond} Rn, Operand 2

eg:- TEQ R1, #10

* R1 is the register to be tested.
* #10 is an immediate value to be xored with value in R1

iii) UMLAL [Unsigned Multiply Accumulate Long]

* It is used to multiply 2 unsigned 32-bit integers & store the 64-bit result in 2 32-bit registers, but it also adds 64-bit result to the value in destination register. Before storing it in RdLo & RdHi

* Syntax :- UMLAL {RdLo, RdHi}, Rn, Rm

* RdLo holds lower 32 bits
  RdHi " Upper " "
  Rn Specifies first operand
  Rm " 2nd "

eg:-

```
ENTRY
START
    ldr R0,=#0x5
    ldr R1, 0x5
    LDR R2, =0x2
    LDR R3, =0x4
    UMLAL R3, R2, R0, R1
BACK B BACK
END
```

Factorial Code

```
ENTRY   MOV R0, #3
        MOV R1, R0
FACT    SUBS R1, R1, #1
        CMP R1, #1
        BE EQUAL
        MUL R3, R0, R1
        MOV R0, R3
        BNE FACT
EQUAL   STOP
        END
```

SUM Code

```
ENTRY
        MOV R1, #10
        MOV R2, #0
LOOP
        ADD R2, R2, R1
        SUBS R1, #0x01
        BNE LOOP
BACK
        END
```