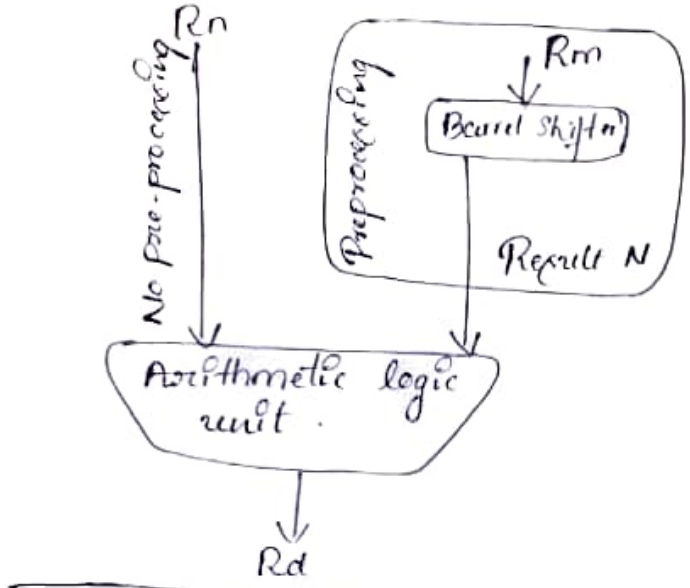
# Arm embedded Systems
## Assignment - 2

1) What is the importance of Barrel shifter Data Processing instructions explain with relevant examples.




Barrel Shifter & ALU.

- N/c can be more than just a register or immediate value.
- It can be a register Rm that has been preprocessed by the barrel shifter prior to being used by a data processing instructions.
- Data processing instructions are processed within the arithmetic logic unit (ALU).
- A unique and powerful feature of the ARM processor is the ability to shift the 32 bit binary pattern in one of the source registers left or right by a specific no of positions before it enters the ALU.
- This shift increases the power & flexibility of many data processing operations.
- Pre-processing or shift occurs within the cycle time of the instruction. This is particularly useful for loading constants into a register & achieving fast multiplies or division by a power of 2.

r0 = 0x00000000
r1 = 0x80000004

MOVS r0, r1, LSL #1

POST cpsr = nzCvqiFt_USER
r0 = 0x00000008
r1 = 0x80000004

2) Different addressing modes for load store multiple instructions with relevant examples.

→

| Addressing mode | Description | Start address | End address | Rn! |
|---|---|---|---|---|
| IA | Increment after | Rn | Rn+4*N-4 | Rn+4*N |
| IB | Increment before | Rn+4 | Rn+4*N | Rn+4*N |
| DA | Decrement after | Rn-4*N+4 | Rn | Rn-4*N |
| DB | Decrement before | Rn-4*N | Rn-4 | Rn-4*N |

* The base register Rn determines the source or destination address for a load store multiple instruction.
* This register can be optionally updated following the transfer.
* This occurs when register Rn is followed by the ! character.

Eg: Register r0 is the base register Rn and is followed by !, indicating that the register is updated after the instruction is executed.

the "-" character is used to identify a range of registers (In this case the range is from register r1 to r3 inclusive)

mem32[0x80010] = 0x01
r0 = 0x00080010
r1 = 0x00000000
r2 = 0x00000000
r3 = 0x00000000

LDMIA r0!, {r1-r3}

POST $r0 = 0X0008001c
$r1 = 0X00000001
$r2 = 0X00000002
$r3 = 0X00000003

3) SWP instruction and Semaphore implementation example.

→ • The Swap instruction is a special case of a load store instruction. It swaps the contents of memory with the contents of a register.
• This instruction is an atomic operation.
• It reads & writes a location in the same bus operation, preventing any other instruction from reading or writing to that location until it completes.

Syntax: SWP{B}{<cond>} Rd, Rm, [Rn]

Eg: Swap instruction loads a word from memory into register r0 & overwrites the memory with register r1.

PRE mem32[0X9000] = 0X12345678
$r0 = 0X00000000
$r1 = 0X11112222
$r2 = 0X00009000

SWP $r0, $r1, [$r2]

POST mem32[0X9000] = 0X11112222
$r0 = 0X12345678
$r1 = 0X11112222
$r2 = 0X00009000

Eg: Shows a simple data guard that can be used to protect data from being written by another task. The SWP instruction "holds the bus" until the transaction is complete.

```
spin
    MOV   r1, "Semaphore
    MOV   r2, #1
    SWP   r3, r2, [r1] ; holds the bus until complete
    CMP   r3, #1
    BEQ   spin
```

- The address pointed by the semaphore either contains the value 0 or 1
- When the semaphore equals 1, then the service in question is being used by another process.
- The routine will continue to loop around until the service is released by the other process - in other words, when the semaphore address location contains the value 0.

4) Atomic operation: ARM says that Load & Store instructions are atomic & it's execution is guaranteed to be complete before interrupt handler executes.

5) Explain the working of Stack Memory considering STMFD and STMED.

5) Explain Push and Pop instruction in Thumb stack operations with examples.

→ The ~~stack~~ thumb stack operations are different from the equivalent ARM instructions because they use the more traditional POP & PUSH concept

Syntax: POP {low_register_list {, pc}}
PUSH {low_register_list {, lr}}

| | | |
|---|---|---|
| POP | pop registers from the stack | $Rd^{*N} <-- mem32[sp+4*N], sp = sp+4*N$ |
| PUSH | push registers on to the stack | $Rd^{*N} --> mem32[sp+4*N], sp = sp-4*N$ |

- There is no stack pointer in the instruction. because the stack pointer is fixed as register R13 in Thumb operations & SP is automatically updated.
- The list of register is limited to the low registers R0 to R7 (link register)
- The PUSH register list is also can include the LR & POP include PC.
- This provides support for subroutine entry & exit.

Eg: To use the POP & PUSH instructions
The subroutine ThumbRoutine is called using a branch with link (BL) instruction.

```
; Call subroutine
BL   ThumbRoutine
; continue
```

```
ThumbRoutine
        PUSH    {r1, lr}        ; enter subroutine
        MOV     r0, #2
        POP     {r1, pc}        ; return from subroutine
```

* lr is pushed onto the stack with r1
* upon return, register r1 is popped off the stack, as well as the return address being loading into the pc
This returns from the subroutine.

6) Explain Conditional execution in ARM & explain the same by writing an ALP subroutine to find GCD of two numbers.

→ Conditional Execution:

- Most ARM instructions are conditionally executed – the instructions only executes if the condition code flags pass a given condition or test.
- It permits to increase the performance & code density
- The condition field is a two-letter mnemonic appended to the instruction mnemonic.
- The default mnemonic is AL or always execute.
- CE reduces the number of branches & pipeline flushes. thus performance of the executed code is becomes to improved.
- CE depends on 2 components
  1) Condition field
  2) Condition flags

Eg: shows an ADD instruction with the EQ condition appended This instruction will only be executed when the zero flag in the cpsr is set to 1.

```
; r0 = r1 + r2   if zero flag is set
ADDEQ r0, r1, r2
```