UNIT - IV Introduction to ARM Processor

ARM Architecture:-

1st ARM Processor was develop at Acorn Computer LLd of Cambridge, England b/w 1983 & April 1985. At that time until formation of Advanced Risc machines Ltd.

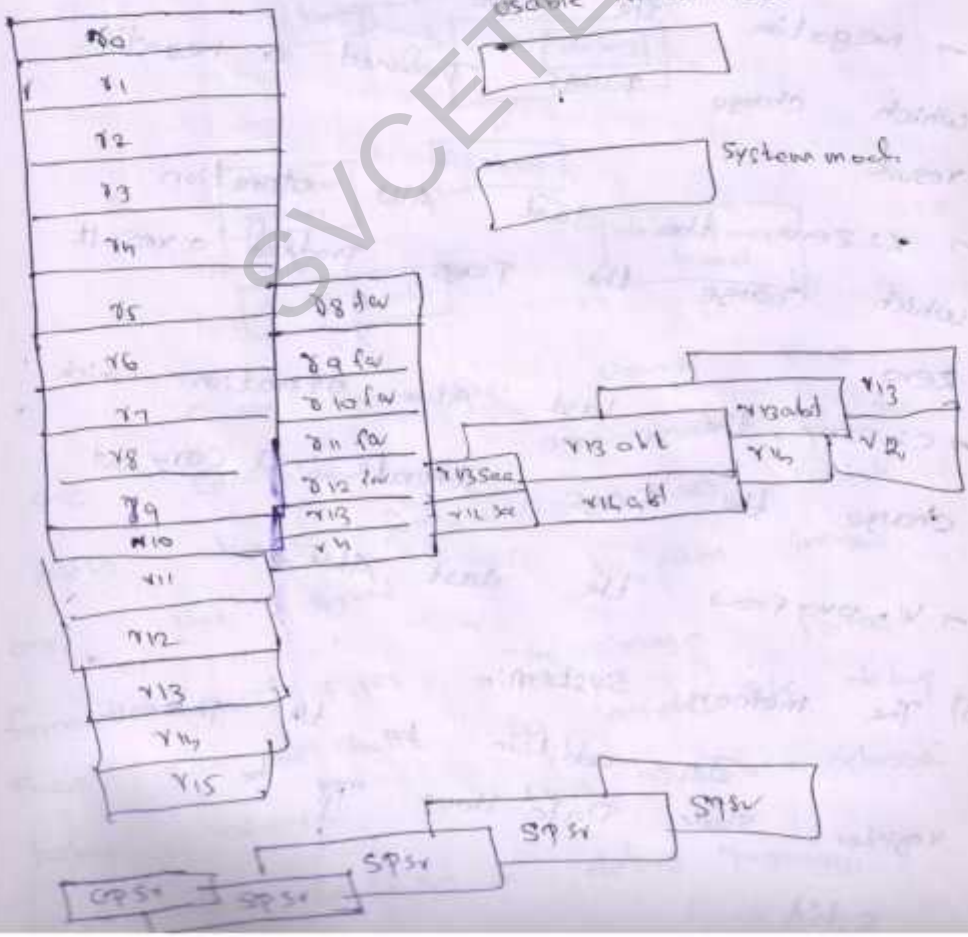
Acorn had develop a Strong position in uk Personal Computer market due to the success of BBC microcomputer.

The BBC micro was a machine 8 bit 6502 microprocessor & rapidly became established.

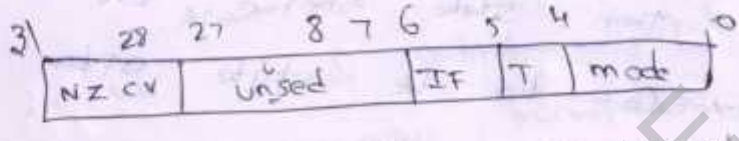
As a result of these frustration with commercial microprocessor offer the design of proprietary microprocessor was considered the major stumbling block way Acorn Learn knew commercial microprocessor It had to produce a better design with fraction of design effect with no experience in custom chip design beyond a few small gate arrays design for the BBC micro

The ARM Programmer model -

A processor instruction set define the operation that programer use to change the state of system incorporated the processor. This state usually comprise the value of data item in processor visible register & system memory. Each instruction can be viewed as performing a defined transformation from state before the instruction execute to the state after it has completed



when writing user level Program only 15 general Purpose 32 bit registers the Program Counter (r15) & current Program Status register (CPSR) considered The remaining registers are used only For System level Program & For handling exceptions



(a) Current Program status register

- negative the last ALU operation which change Flags Produced a negative result
- Z: zero. the last ALU operation which change the Flags produced a result zero.
- C: carry, the last ALU operation which change the Flags generate a carry out
- V - overflow the last ALU

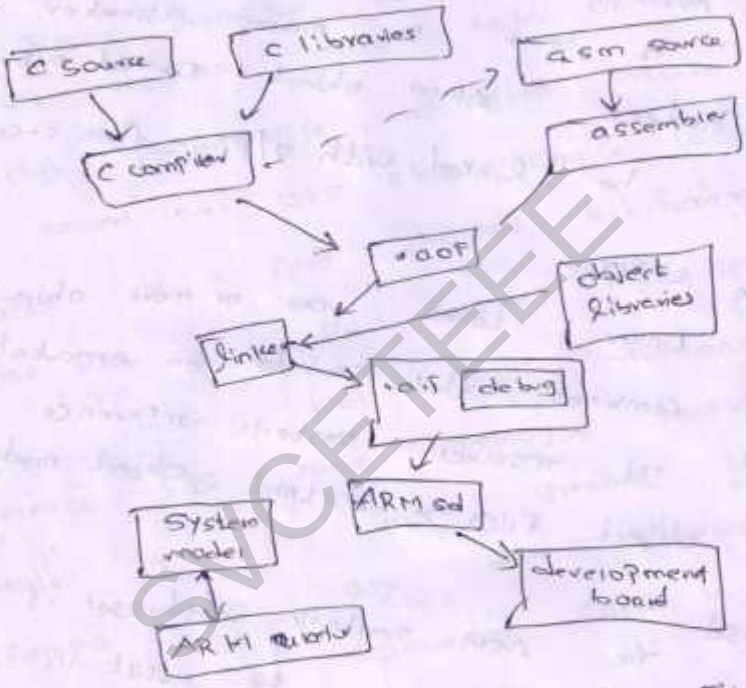
(b) The memory System:-

In addition to the processor register state Data items may be

8-bit

ARM Development Tools:-

ARM is supported by a coherent range of tools develop by ARM Limited. software development for the and there also many third party & public domain tools available such as an ARM back end gcc C compiler



C or assembler source File are Compiler or assembled in to ARM object format (.aof) files, which are then linked in to ARM image format (.axf) files, The image format files can be built to include the debug table required by ARM Symbolic debugger. which can load, run & debug programs.

(a) ARM C - compiler:-

It is complaint with ANSI standard. For C and is support by appropriate library of standard function. It uses the ARM procedure call standard for all extremely available function.

(b) ARM Assembler:

It is a full macro assembly which produces ARM object format o/p that can be linked with o/p from c-compiler

(c) The Linker:-

Linker takes one or more object Files & Combines them in to an executable Program. It resolves symbolic reference b/w object Files & extract object module.

(d) ARMsd

The ARM symbolic debugger is a front-end interface to assist in debugging program running either under emulation or remotely on a target system such as ARM development board. At its most basic, ARMsd allows an executable program to be loaded in to ARM hardware or a board & run

Memory Hierarchy:-

Memory size * speed:

A typical computer memory hierarchy comprise several levels, each level having a characteristic size * speed

→ The Processor register can be view as the top of the memory hierarchy.

A RISC Processor will typically have around 32-bit registers make total of 128 bytes.

* High Performance desktop system may have second level off chip cache with a capacity of a few hundred K bytes and access time of Few tens of nano seconds.

Note that the performance difference b/w main memory & backup storage is very much greater than difference b/w any other adjacent level there is no Secondary cache in system.

(a) memory cost:-

Fast memory is more expensive per bit than slow memory. So a memory hierarchy also aims to give a performance close to the faster memory with an average cost per bit approaching slow memory.

Caches:-

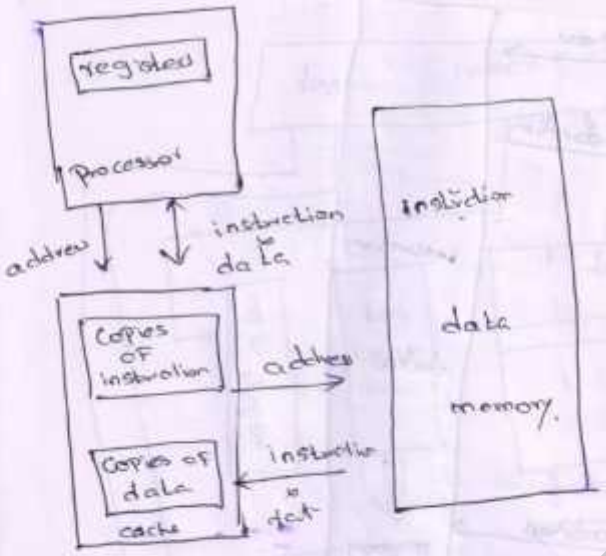
The First RISC Processor were introduced at time when standard memory Parts were Faster than their contemporary micro Processor, but this situation did not Persist for long subsequent advances in Semiconductor Process technology which have been exploited to make micro Processor Faster have been applied differently to improve memory chips

(a) Processor & memory speeds:-

In 1980 a typical DRAM Part could hold 4K bits of data. 16k bit chips arriving in 1981 & 1982 These Parts would cycle at 3 or 4 mHz For random access and about twice the rate of total access

Unified & Harvard caches:-

caches can be built in levels a At highest may ways: can have one of Following Processor two organization



Cache Performance Metrics:-

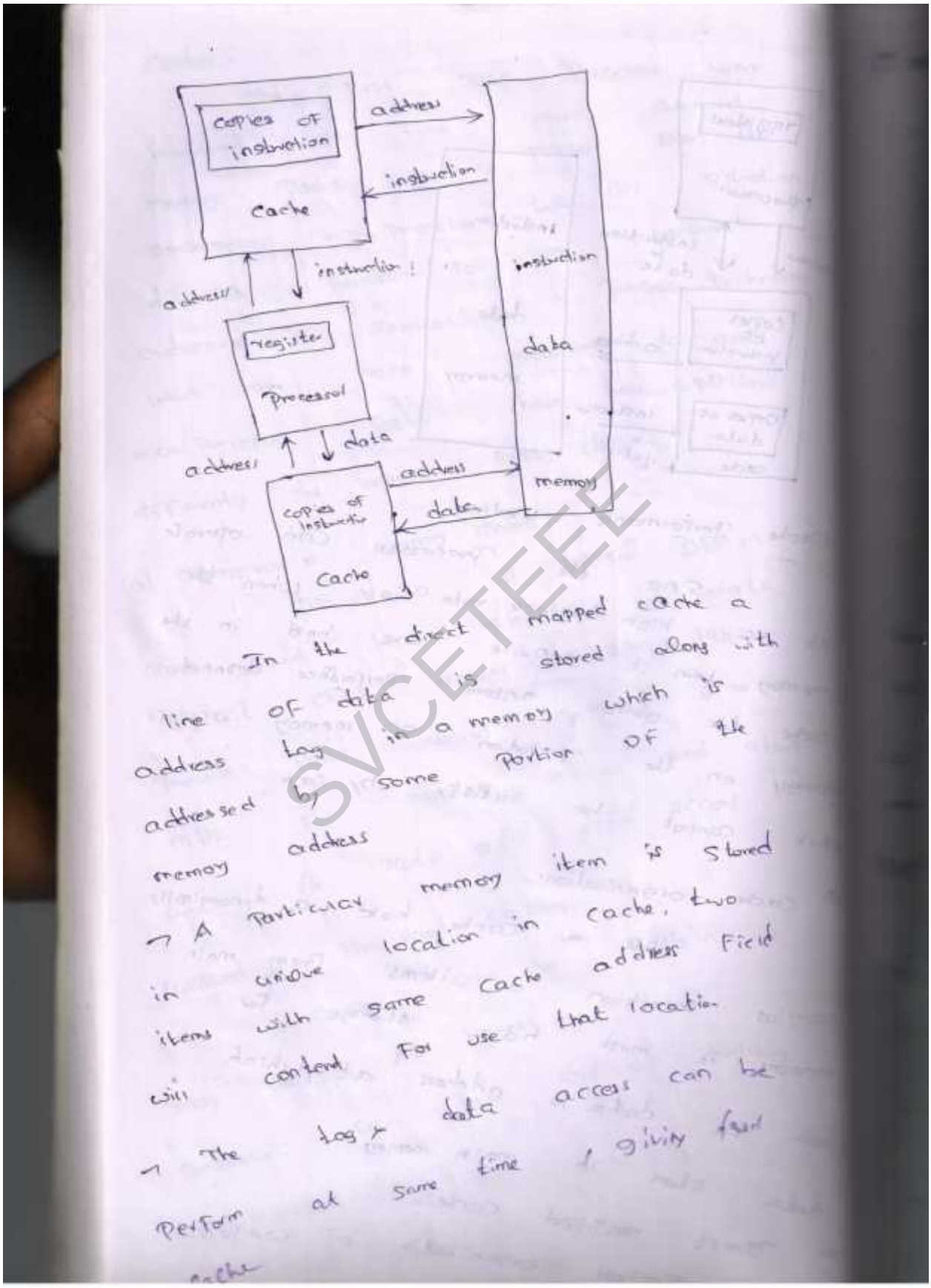
Since the Processor can operate at its high clock rate only when the memory item it requires are held in the cache the overall system performance depend strongly on the proportion of memory accesses which cannot be satisfy by cach

(d) Cache organization:-

Since a cache hold a dynamically varying selection of items from main memory it must have storage for both the data & address at which data store in main memory

(a) Direct mapped cache:-

Cache organization of component

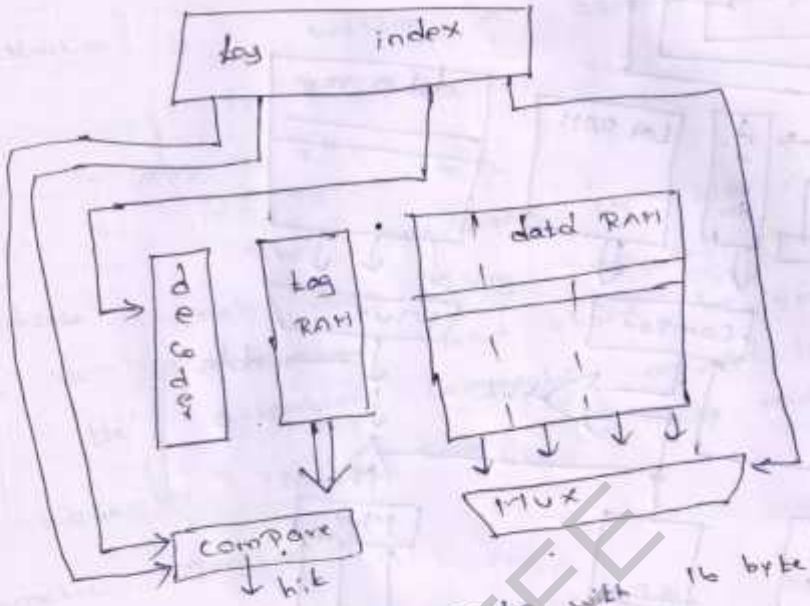


In the direct mapped cache a line of data is stored along with address tag which is a memory address addressed by some portion of the memory address

→ A particular memory item is stored in unique location in cache. two items with same cache address field will content For use that locatio.

→ The tag + data access can be Perform at same time, giving fast

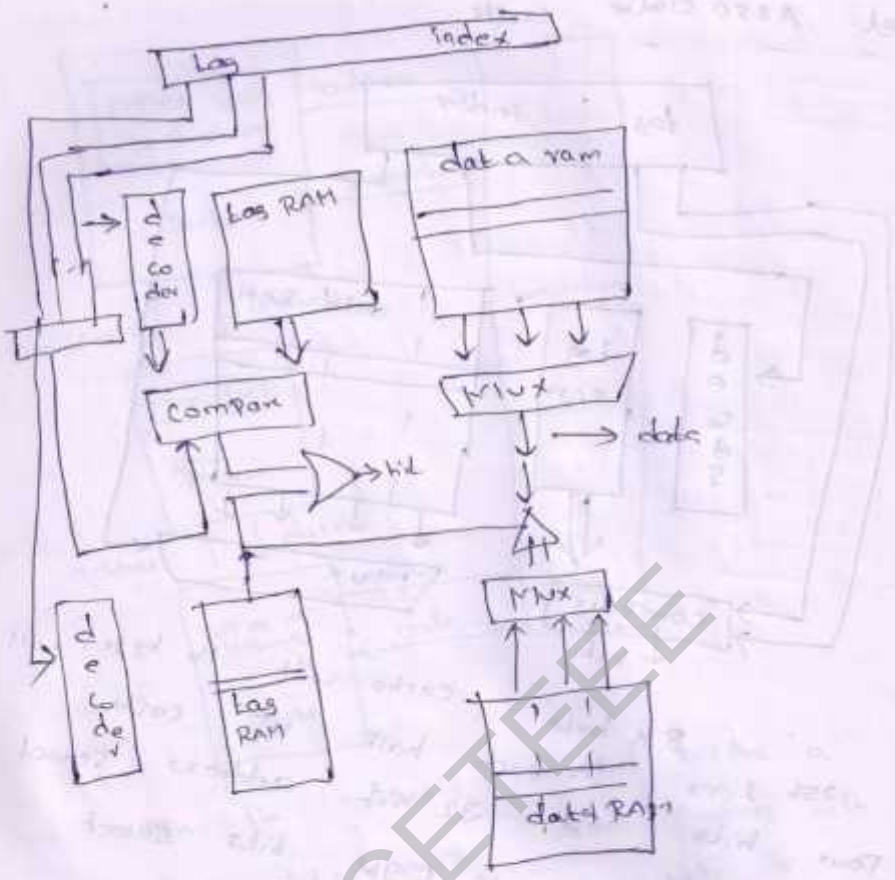
(M) set Associative Cache



8 K byte cache with 16 byte will be 256 lines in each half of cache
So four bits of 32 bit address select a byte from line to eight bits select one line from each half of cache.

Random allocation:-
The decision is based on random or pseudo random value.

Least recently used:-
Cache keep a record of which pair was last accessed location to other one.
allocate new data
Round - Robin also known as cyclic



ARM Assembly Language Programming

Data Processing instruction:-

ARM data processing instruction enable the programmer to perform arithmetic & logical operation on data value of register.

* All operands are 32 bits wide & from registers or specified
* The result is then is one in 32 bits wide & is placed in with

(a) Simple Register operands

A typical ARM data processing instruction is written in assembly language

```
r0, r1, r2
ADD r0, r1, r2 ; r0 := r1 + r2
```

The semicolon in this line indicate that every thing to its right of it is comment and should be ignored by the assembler. comments are put in to assembly source code to make reading.

Arithmetic operation:-

ADD	r0 r1, r2 ;	r0 = r1 + r2
ADC	r0 r1, r2 ;	r0 = r1 + r2 + C
SUB	r0 r1, r2 ;	r0 = r1 - r2
SBC	r0 r1, r2 ;	r0 = r1 - r2 + C - 1
RSB	r0 r1, r2 ;	r0 = r2 - r1
RSC	r0 r1, r2 ;	r0 = r2 - r1 + C - 1

Bit wise logical operation:-

These instruction perform the specified logic operation one each bit pair of the i/p operand

AND	r0 r1, r2 ;	r0 = r1 & r2
ORR	r0, r1, r2 ;	r0 = r1 or R2
EOR	r0 r1, r2 ;	r0 = r1 X or r2
BIC	r0 r1, r2 ;	r0 = r1 & not r2

Comparision operation:

→ These instruction do not produce a result therefore omitted from the assembler language Format.

→ but just the condition code bits (N, Z, C & V) CPSR according to select operation)

CMP CMN
r_1, r_2 : set cc on $r_1 - r_2$
r_1, r_2 : Set cc on $r_1 + r_2$
r_1, r_2 : Set cc on r_1 bit r_2
r_1, r_2 : Set cc on r_1 x or r_2

(b) Immediate operands:-

IF instead of add two registers we simply wish to add a constant to register we can replace the second source operand with an immediate value which is literal constant, preceded by #

ADD r_3, r_3 ; $r_3 := r_3 \neq 1 + 1$
AND r_8, r_7 ; $r_2 := \neq FF$ r7

The first example also illustrate the although 3 address format also allow source

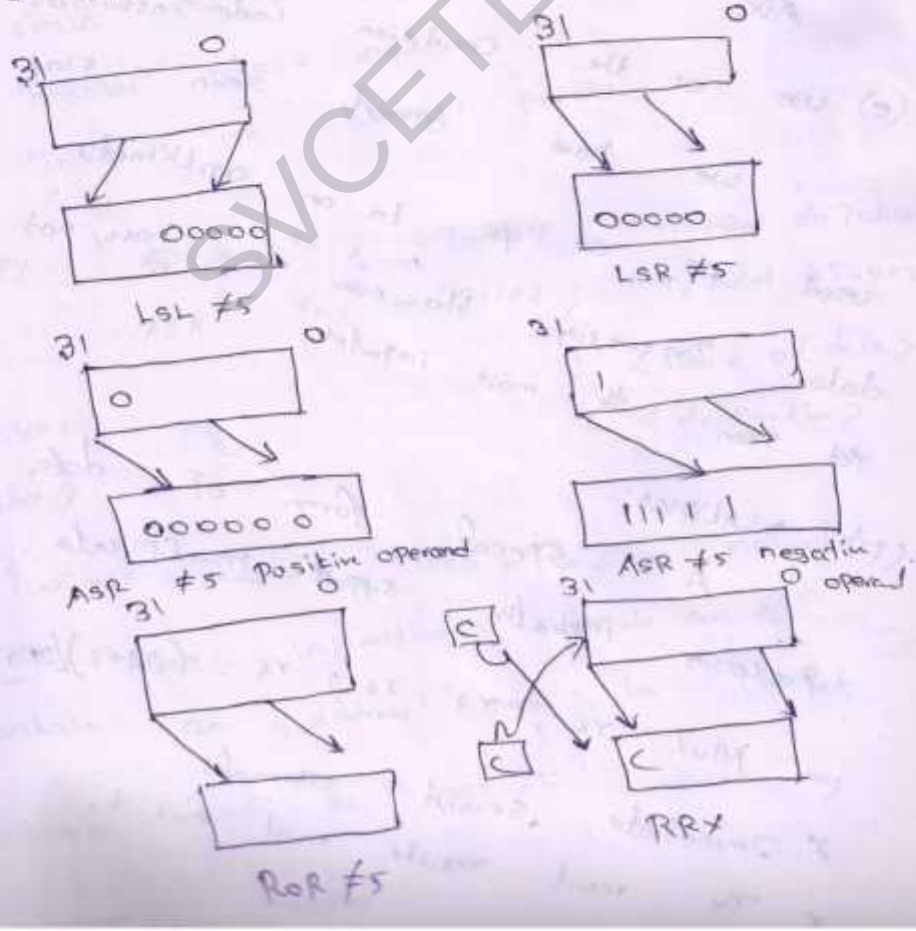
(c) Shifted Register operands :-

A third way to specify a data operation is similar to first but allows the second register operand to be subject to shift operation

ADD r3, r2, r1 LSL #3 r3 := r2 + 8 × r1

LSL : logical shift left by 0 to 31 places; Fill the vacated bits at least significant end of word with zero

LSR :- Logical shift right by 0 to 32 places; Fill the vacated bits at most significant end of word with zeros



ADD r5 r5 r3 LSL r2 r2 r4

(d) setting the condition codes:-

Any data Processing instruction can set the Condition codes if the Programmer wishes it to. The compilers operation only set the condition codes. So there is no option with them, but for all data processing instruction. a specific request can be made.

ADDS r2 r2 r0 : 32 bit carry out added in the high word

ADC r2 r3 r1

(e) use of the Condition code Instruction:

we have already seen 'c' flag used as an i/p to an arithmetic data processing. we have not yet seen the most important

(f) Multiplexi-

A special form of data processing instruction support multiplication

MUL r4, r3, r2; r5 = (r3 r2)[31:0]

* Immediate second operands
* The result register must not be

Multiply two 32-bit register gives 64 bit result level significant 32 bit of which are placed in result register this can be viewed as multiplication in module

MLA R4, R3, R2, R1 ; R4 := (R3 x R2 + R1)

Data Transfer instruction:

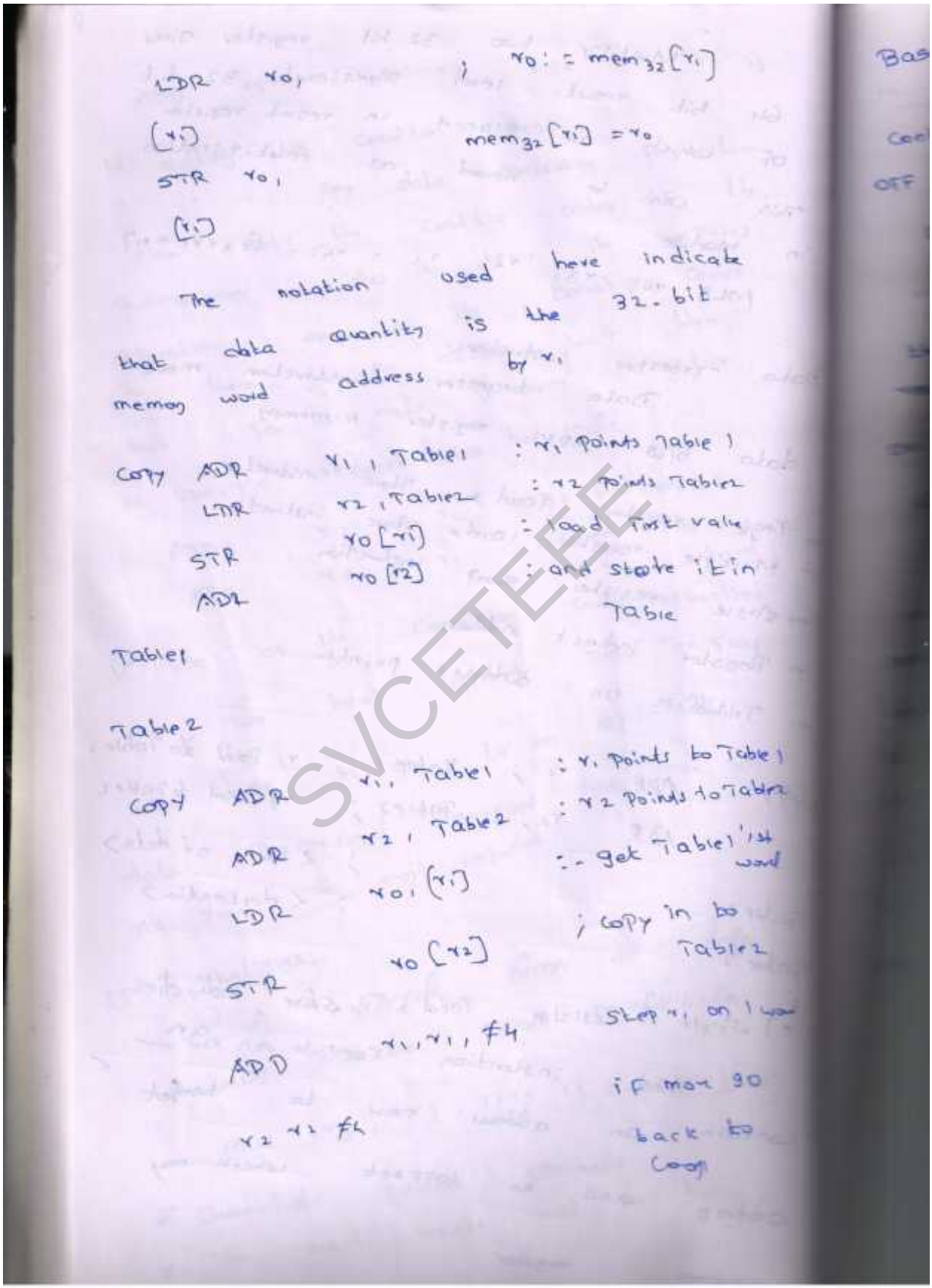
Data transfer instruction move data b/w ARM register & memory.

- Single register load & store instruction
- Multiple register load & store instruction
- single register swap instruction
- Register indirect Addressing
- Initialize an address pointer

```
COPY    ADR R1 ; Table 1 ; R1 Point to Table 1
        ADR R2 ; Table2 ; R2 Point to Table2
                ; <source of data>
                : <destination>
Table 1
Table 2
```

(c) Single Register load & store instruction

These instruction compute an effective address by adding an address register contain an address near to target address and an offset which may



LDR r0, [r1] ; r0 := mem32[r1]

STR r0, [r1] mem32[r1] = r0

The notation used here indicate that data quantity is the 32-bit memory word address by r1

```
Copy  ADR  r1, Table1   ; r1 Points Table1
      LDR  r2, Table2   ; r2 Points Table2
      STR  r0 [r1]      ; Load Test value
      ADR  r0 [r2]      ; and store it in
                          Table

Table1

Table2

Copy  ADR  r1, Table1   ; r1 Points to Table1
      ADR  r2, Table2   ; r2 Points to Table2
      LDR  r0, [r1]     ;- get Table1's 1st word
      STR  r0 [r2]      ; Copy in to Table2
      ADD  r1, r1, #4   Step r1 on 1 word
      ADD  r2 r2 #4     if more go back to Copy
```

Base plus offset Addressing:-

If base register does not contain exactly the right address an offset up to 4k bytes may add

LDR r0, r0 = mem32 [r1 ≠4] +4

The exclamation mark indicate the instruction should update the base register after initiating data transfer on the ARM is this auto index

LDR r0, [r1] ≠4 r0 = mem 32 [r1]
r1 = r1 + 4.

Here the exclamation mark is not needed, since the only of the immediate offset is as a base register modifier. Again this offset the instruction is exactly equivalent to a simple register indirect load. Follow by a data copy

ADR r1, Table1 ; r1 points to Table1
ADR r2, Table2 ; r2 points to Table 2.

Loop LDR [r1] ; get table 1st word
STR r0 [r2]; ≠4 ; copy to table2
; if more go back to loop

Table 1 ;<Source of data>

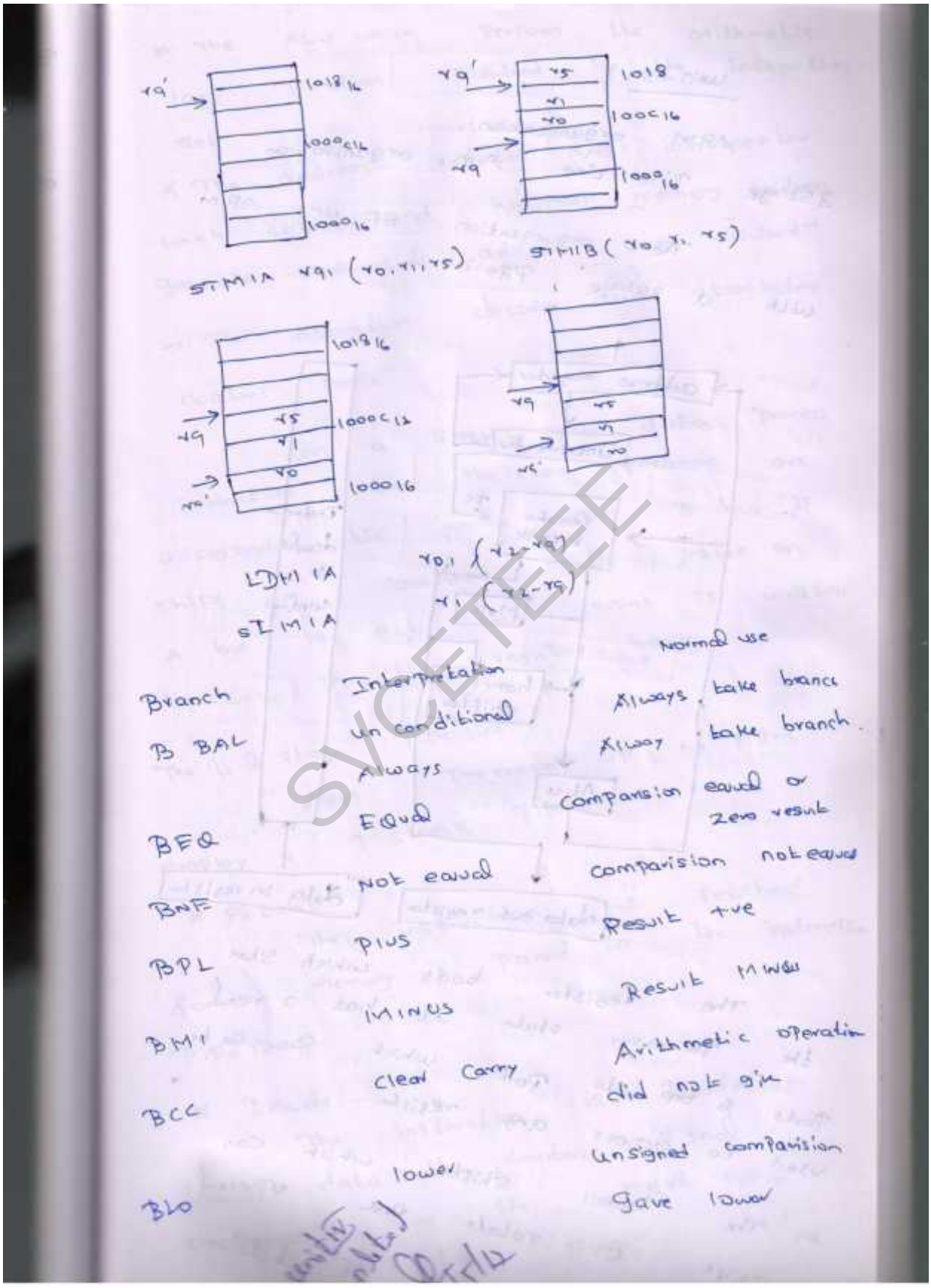
Table 2 : <destination>

(9) Block copy Addressing

Although the stack view of multiple register transfer instruction is useful there are occasion when a different view is easier to understand. For example when the instruction are used to copy a block of data The mapping b/w two views depend on whether the operation is a load or store.

	Ascending		Descending	
	Full	Empty	Full	Empty
Before	STMIB			LDMIB
Increment	STMFA			LDMED
After		STMIA	STMEA	LDMID
		STMEA		LDMFD
Before		LDMDB	STMDB	
		LDMEA	STMFD	
Decrement	LDMDA			STMDA
After	LDMFA			STMFD

The block copy views are illustrated.


STMIA R9! (R0, R1, R5)
STMIB (R0, R1, R5)
LDMIA
STMIA
R0! (R2-R0)
R1! (R2-R5)
Branch
Interpretation
Normal use
B BAL
Unconditional
Always
Always take branch
Always take branch
BEQ
Equal
Comparision equal or zero result
BNE
Not equal
Comparision not equal
BPL
Plus
Result +ve
BMI
Minus
Result minus
BCC
Clear Carry
Arithmetic operation did not give
Unsigned comparision gave lower
BLO
lower