classmate
Date 05/01/2021
Page 1

L-T-P-C : 3-0-3-4

COURSE OBJECTIVES:

1. To introduce the advanced features of an advanced RISC ~~microcontroller~~ Microprocessor.
2. To apply the knowledge of embedded C Programming for configuring various peripherals of a microcontroller.
3. To design and develop Microcontroller based solution for solving real world problems.

COURSE OUTCOMES:

CO1: Able to identify the importance of 32 bit Microprocessor.
CO2: Able to understand architecture of the ARM processor.
CO3: Able to analyze peripherals and their programming aspects.
CO4: Able to design and develop embedded systems using microcontroller.

Q. What is an Embedded system?
A. System in which software (firmware) is embedded into the hardware. The core part of the system will be a programmable device (microprocessor / microcontroller / etc.)
Examples -

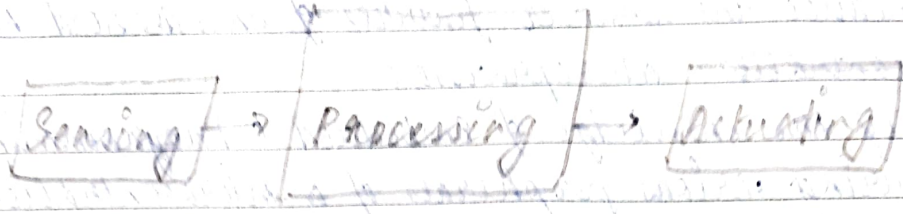
1. Avionics
2. Communication
3. Automobile
4. Consumer Electronics
5. Office Equipments
6. Household Appliances

AUTOMOTIVE EMBEDDED SYSTEMS:

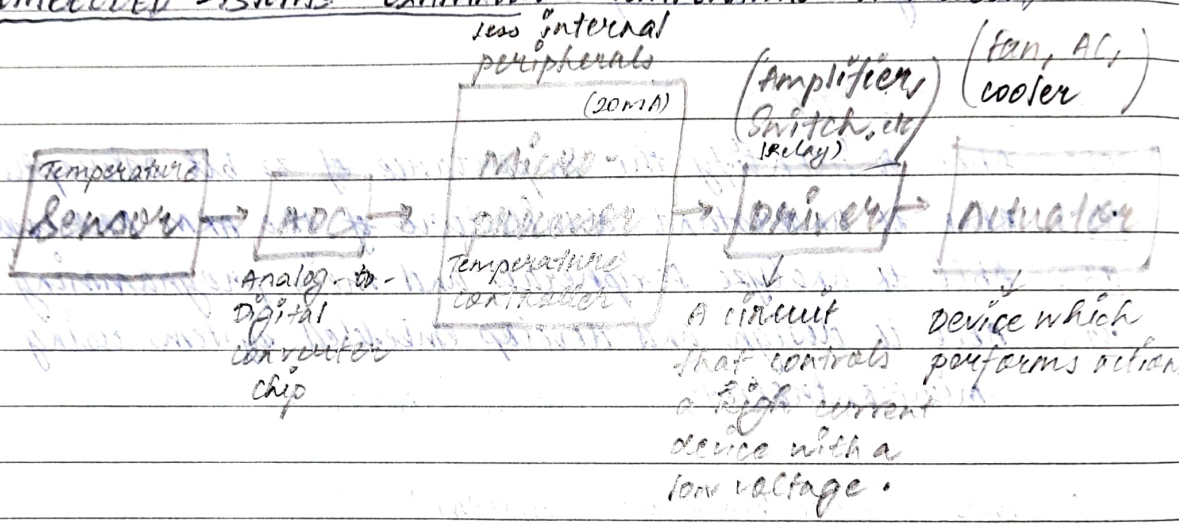
Today's high-end automobile have >80 microprocessors:

1. 4-bit microcontroller checks seat belt;
2. microcontrollers run dashboard devices;
3. 16/32-bit microprocessor controls engine.
4. Millions lines of code

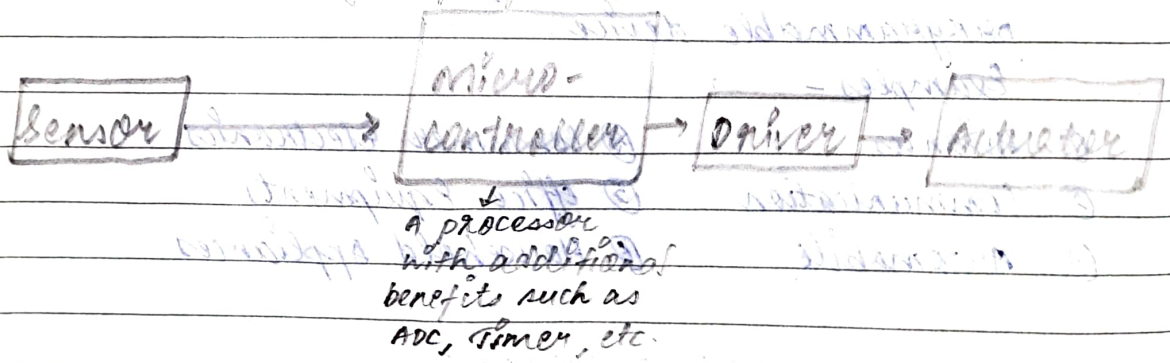
* OPERATIONS - EMBEDDED SYSTEMS :



* EMBEDDED SYSTEMS - EXAMPLE : (TEMPERATURE CONTROLLER)



Replacing micro-processor with a micro-controller,



* MICROPROCESSOR :

① Requires 'external' support hardware
Example - External RAM, ROM, Peripherals

② Application : Processing - Arithmetic, Logic Operations.

* MICROCONTROLLER :

1. Very little external support hardware / stand alone.
2. Most RAM, ROM and peripherals on chip.
3. "Computer on a chip", or "System on chip" (SOC)
   Example - PSC (Peripheral Interface Controller)
4. Application : Controlling purposes.

Q. Why Embedded Systems ?

A. 1. Reduced number of components.
2. Reduced size.
3. Reduced cost.
4. Reduced power consumption.
5. Easier upgradation.
6. Easier troubleshooting & maintenance.
7. Best suited for specific controlling applications.

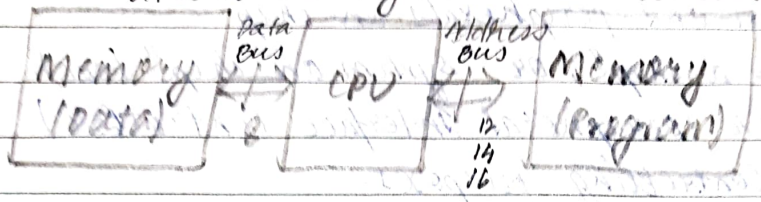
* NOTE : The CPU of a microcontroller contains Oscillator (0-40 MHz), A/D Converter, Microprocessor, RAM, and Program Memory.
[Refer to slide for diagrammatic representation.]

* VARIOUS MICROCONTROLLERS :

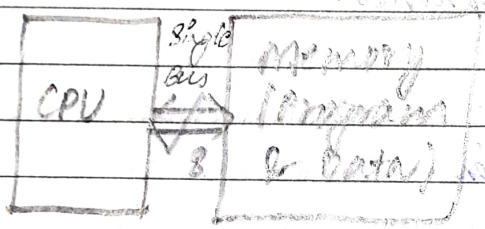
1. 8 bit microcontrollers
   (i) Microchip - PIC 12 & 16 series.
   (ii) Atmel - 89c51
   (iii) Intel - 8051
   (iv) Motorola - 68HCxx series
2. 16 bit microcontrollers
   (i) Microchip - PIC 18 series.
3. 32 bit microcontrollers
   (i) ARM processors
4. DSP based microcontrollers
   (i) Shark

TWO DIFFERENT ARCHITECTURES :

BUS : Path through which interaction takes place between memory and CPU simultaneously [collection of signals]



Harvard Architecture



Von-Neumann Architecture

	VON-NEUMANN ARCHITECTURE	HARVARD ARCHITECTURE
(1)	It is ancient computer architecture based on stored program computer concept.	It is modern computer architecture based on Harvard Mark I relay based model.
(2)	Same physical memory address is used for instructions and data.	Seperate physical memory address is used for instructions and data.
(3)	There is common bus for data and instruction transfer.	Seperate buses are used for transferring data and instruction.
(4)	Two clock cycles are required to execute single instruction.	An instruction is executed in a single cycle.
(5)	It is cheaper in cost.	It is costly than Von Neumann Architecture.
(6)	CPU cannot access instructions and read/write at the same time.	CPU can access instructions and read/write at the same time.

①	It is used in personal computers and small computers.	It is used in micro controllers and signal processing.

* RISC vs. CISC PROCESSOR :

	COMPLEX INSTRUCTION SET COMPUTER (CISC)	REDUCED INSTRUCTION SET COMPUTER (RISC)
①	A large number of instructions are present in the architecture. (usually >100)	Very fewer instructions are present. The number of instructions are generally less than 100 (usually <50)
②	Some instructions with long execution times. These include instructions that copy an entire block from one part of memory to another and others that copy multiple registers to and from memory.	No instruction with a long execution time due to very simple instruction set. Some early RISC machines did not even have an integer multiply instruction, requiring compilers to implement multiplication as a sequence of additions.
③	Variable-length encodings of the instructions. Example - IA32 instruction size can range from 1 to 15 bytes.	Fixed-length encodings of the instructions are used. Example - In IA32, generally all instructions are encoded as 4 bytes.
④	Multiple formats are supported for specifying operands. A memory operand specifier can have many different combinations of displacement, base and index registers.	Simple addressing formats are supported. Only base and displacement addressing is allowed.
⑤	CISC supports array. Several addressing modes	RISC does not supports array. Only a few addressing modes.

(6)	Arithmetic and logical operations can be applied to both memory and register operands.	Arithmetic and logical operations only use register operands. Memory referencing is only allowed by load and store instructions, i.e. reading from memory into a register and writing from a register to memory respectively.
(7)	Implementation programs are hidden from machine level programs. The ISA provides a clear abstraction between programs and how they get executed.	Implementation programs exposed to machine level programs. Few RISC machines do not allow specific instruction sequences.
(8)	Condition codes are used. The stack is being used for procedure arguments and return addresses.	No condition codes are used. Registers are being used for procedure arguments and return addresses. Memory references can be avoided by some procedures.
(9)	Usually takes more than 1 internal clock cycle (Tcyc) to execute.	Executes 1 instruction in 1 internal clock cycle (Tcyc)
(10)	Used in : 80x86, 8051, 68HC11, etc.	Used in : SPARC, ALPHA, Atmel AVR, etc.

* DIFFERENCE BETWEEN MICROPROCESSOR AND MICROCONTROLLER :

PTO

MICROPROCESSOR	MICROCONTROLLER
① Microprocessor is the heart of computer system. It is only a processor, so memory and I/O components need to be connected externally.	Microcontroller is the heart of an embedded system. It has a processor along with internal memory and I/O components.
② Memory and I/O has to be connected externally, so the circuit becomes large.	Memory and I/O are already present, and the internal circuit is small.
③ You cannot use it in compact systems. Cost of the entire system is high.	You can use it in compact systems. Cost of the entire system is low.
④ Due to external components, the total power consumption is high. Therefore, it is not ideal for the devices running on stored power like batteries.	As external components are low, total power consumption is less. So it can be used with devices running on stored power like batteries.
⑤ Most of the microprocessors do not have power saving features. It is mainly used in personal computers.	Most of the microcontrollers offer power-saving mode. It is used mainly in a washing machine, MP3 players, and embedded systems.
⑥ It has a smaller number of registers, so more operations are memory-based.	It has a more number of registers, so the programs are easier to read-write.
⑦ It is based on Von-Neumann model. It is a central processing unit on a single silicon-based integrated chip.	It is based on Harvard architecture. It is a byproduct of the development of microprocessors with a CPU along with other peripherals.
⑧ It has no RAM, ROM, I/O units, timers and other peripherals on the chip.	It has a CPU along with RAM, ROM, and other peripherals embedded on a single chip.

(9)	It uses an external bus to interface to RAM, ROM, and other peripherals.	It uses an internal controlling bus.
(10)	Microprocessor-based systems can run at a very high speed because of the technology involved.	Microcontroller-based systems run up to 200 MHz or more depending on the architecture.
(11)	It is used for general purpose applications that allow you to handle loads of data.	It is used for application-specific systems.
(12)	It is complex and expensive, with a large number of instructions to process.	It is simple and inexpensive with less number of instructions to process.

LECTURE 2 - INTRODUCTION TO ARM PROCESSOR

→ Acron RISC Machines

★ BRIEF HISTORY OF ARM:

(1) ARM was developed at Acron Computer Limited of Cambridge, England between 1983 and 1985.
(2) RISC concept introduced in 1980 at Stanford & Berkley.
(3) ARM Limited founded in 1990.
(4) ARM Cores -
→ Advanced RISC Machines
→ Intellectual property core (IPC)
(i) licensed to partners to develop and fabricate new micro-controllers
(ii) Soft-cores (.bit file)

★ FEATURES OF BASIC RISC ARCHITECTURE:

(1) A large Uniform Register File → collection of registers - same size (32 bit)
(2) Load-Store Architecture, where data processing operates on register content only.
(3) Uniform and fixed length Instructions 32-bit processor
(4) Instructions are 32-bit long

→ Assembly language Program
Advanced C programming

⑤ Good Speed / Power Consumption Ratio
⑥ High Code Density - Data in a unit area

* PRINCIPLE FEATURES / ENHANCEMENTS IN ARM :
① Control over ALU and Shifter (Barrel) for every data processing operations to maximize their usage. → multiple left/right shift in a single cycle
② Auto-Increment and Auto-Decrement Addressing modes to optimize program loops. → methods to read/write data
③ Load and Store Multiple (LDM) (STM) Instructions to maximize data throughput. (LDR) (STR)
④ Conditional Execution of Instructions to maximize Execution throughput.

* ARM ARCHITECTURE VERSIONS :
① Version 1 (1983-1985) - 26 bit Addressing, no Multiply or Coprocessor
② Version 2 - Includes 32-bit result multiply coprocessor
③ Version 3 - 32-bit Addressing
④ Version 4 - Add Signed, Unsigned Half-word & Signed Byte load & Store Instructions
⑤ Version 4T - 16-bit Thumb - compressed form of Instructions
⑥ Version 5T - Superset of 4T adding new Instructions
⑦ Version 5TE - Add. Signal Processing Extension

Examples -
ARM6 : v3
ARM 7 : v3
ARM 7TDMI : v4T (LPC 2148)
StrongArm : v4
ARM 9E-S : v5TE

T: Thumb instruction support (16-bit) - Compressed ARM [subset of ARM instructions]
D: On-chip debug support
M: Enhanced multiplier
I: Embedded ICE hardware (In circuit emulator JTAG)
T2: Thumb-2
S: Synthesizable code
E: Enhanced DSP instruction set
J: Java support, Jazelle
Z: Should be Trustzone?
F: Floating point unit
H: Handshake, clockless design for synchronous or asynchronous design

* ARM vs. THUMB COMPARISON:

	ARM	THUMB
①	32-bit instruction set	16-bit instruction set
②	3-data address instructions	2-data address instructions
③	16 general purpose registers	8 general purpose registers
④	more regular binary encoding (Better performance ↑)	Greater code intensity.

NOTE - (Thumb) If used correctly, can lead to better performance / power-efficiency.

* OVERVIEW - CORE DATA PATH:

① Data Items are placed in register file.
② No Data Processing Instructions directly manipulate Data in memory.
③ Instructions Typically use two Source Registers and single result or Destination Registers
④ A Barrel Shifter on the date path can pre-process data before it enters into ALU.

Indicates that the normal register used by User or System mode has been replaced by an alternative register specific to the exception mode -> Banked Registers

classmate
Date
Page 11

(8) Increment/decrement logic can update the Register content for sequential access independent of ALU.

* BASIC ARM ORGANIZATION:

[Refer to slide 9 for diagrammatic representation.]

LECTURE 3 - ARM PROGRAMMER'S MODEL

ARM REGISTER ORGANIZATION

* ARM REGISTER ORGANIZATION: Total - 37 Registers [17 + 20]

↳ General purpose registers

↳ Banked: Available only in corresponding mode of operation

[Refer to slide 2 for tabular representation.]

Six operating modes

User	Supervisor	Abort	Undefined	Interrupt (normal)	Fast interrupt
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8_fiq
R9	R9	R9	R9	R9	R9_fiq
R10	R10	R10	R10	R10	R10_fiq
R11	R11	R11	R11	R11	R11_fiq
R12	R12	R12	R12	R12	R12_fiq
R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
R15	R15	R15	R15	R15	R15
CPSR					
SPSR	SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

special functions

CPSR: Current Program Status register ↑ 14 general purpose register

stores the status of arithmetic/logic instructions, e.g. carry/c bit

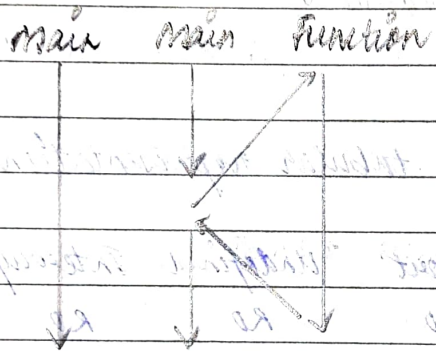
SPSR: Saved Program Status Register → modifies data within the corresponding mode of operation.

* SPECIAL REGISTERS :

1. R13 - Stack Pointer
2. R14 - Link Register
3. R15 - Program Counter

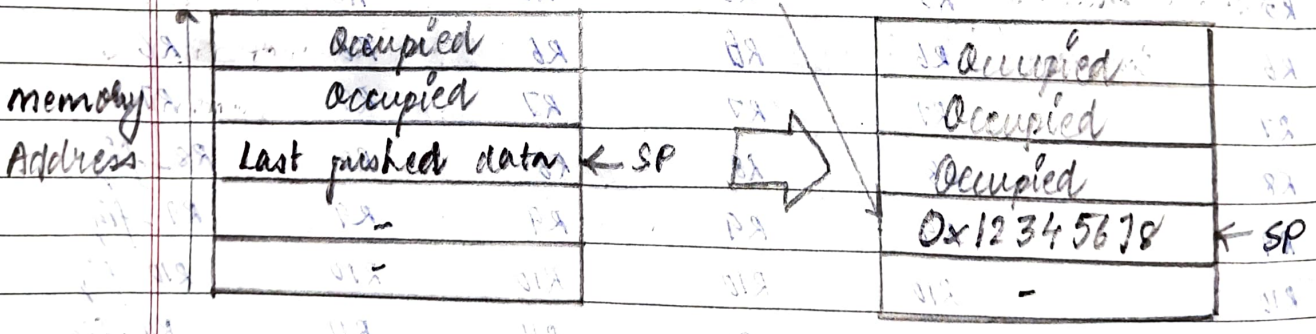
* R13 - STACK POINTER :

Stack Pointer points to Top of stack



R0 | 0x12345678

PUSH {R0}

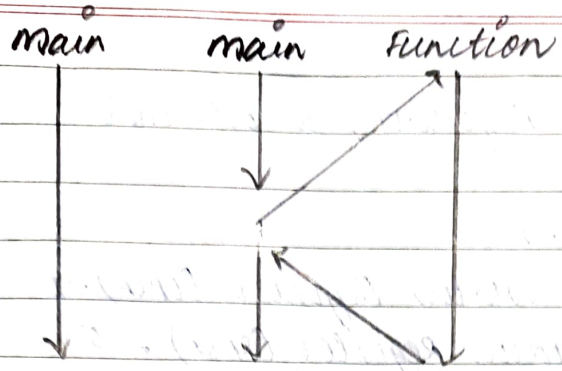


Holds the data where the next pointer has to be pushed on to.

* R14 - LINK REGISTER :

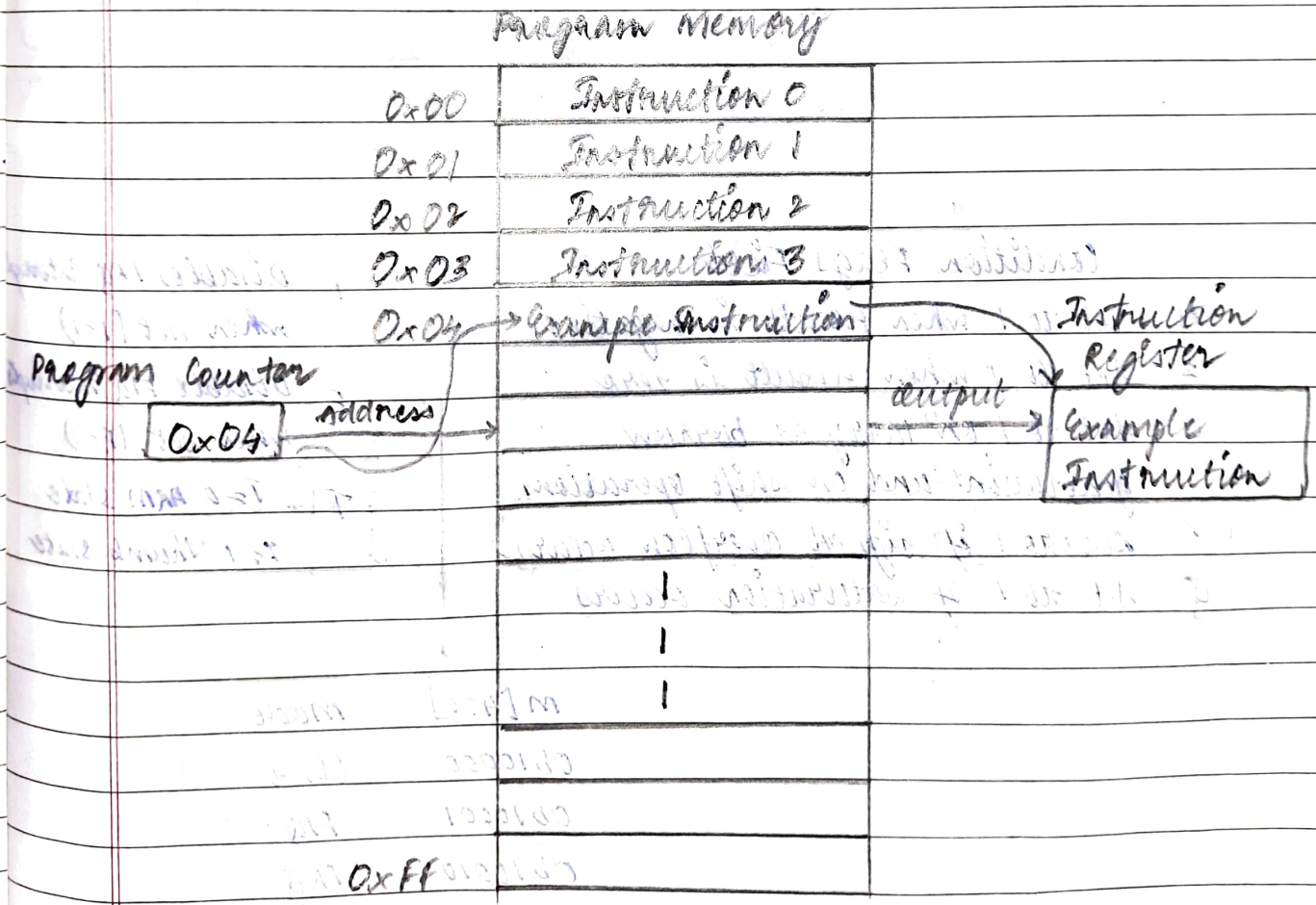
Links Register stores the return Address

PTO



* R15 - PROGRAM COUNTER :

Program Counter points to next Instruction that is to be executed by the Processor



PTO

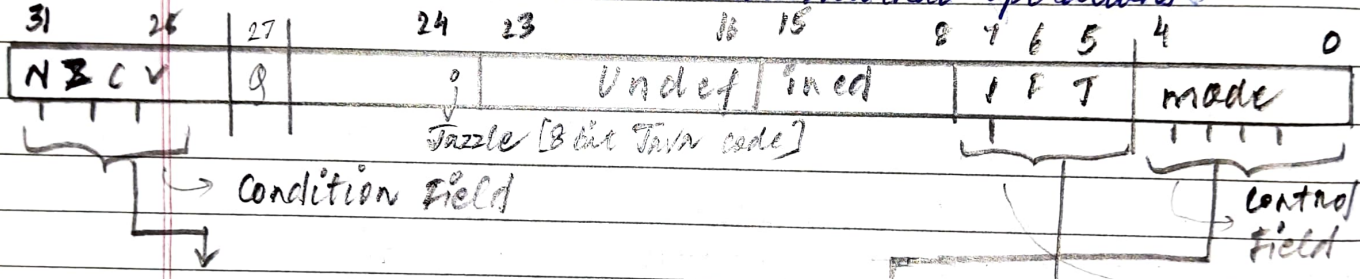
★ ARM REGISTER SETS

ARM has total 37 Registers, 32 bits each

1. Program Counter - 1
2. Current Program status Register (CPSR) - 1
3. Saved Program status Register (SPSR) - 5
4. General Purpose Registers - 30

★ PROGRAM STATUS REGISTER :

CPSR - monitors and control Internal Operations



	Condition Flags Field
N	set to 1 when result is negative
Z	set to 1 when result is zero
C	set to 1 on carry or borrow generation and on shift operations
V	set to 1 if signed overflow occurs
Q	set to 1 if saturation occurs

I	Disables IRQ interrupt when set (I=1)
F	Disable FIQ interrupt when set (F=1)
T	T=0 ARM State
	T=1 Thumb state

M[4:0]	mode
0b10000	User
0b10001	FIQ
0b10010	IRQ
0b10011	Supervisor
0b10111	Abort
0b11011	Undefined
0b11111	System

b - binary

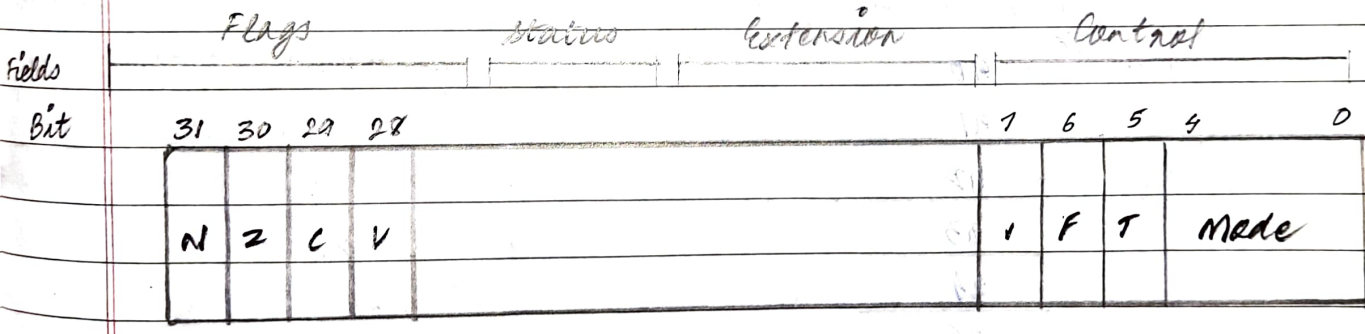
LECTURE 4 - ARM PROCESSOR MODES

* PROCESSOR MODES :

→ Processor Modes determine -
(i) Which registers are active
(ii) Access Rights to CPSR Register itself

→ Privileged -
(i) Full Read-Write access to the CPSR

→ Non-Privileged -
(i) Only Read access to the control Field of CPSR
(ii) Read-Write access to the Condition Flags



ARM has Seven Processor Modes.

Privileged -
(i) Abort
(ii) Fast Interrupt Request (FIQ)
(iii) Interrupt Request (IRQ)
(iv) Supervisor
(v) Undefined
(vi) System

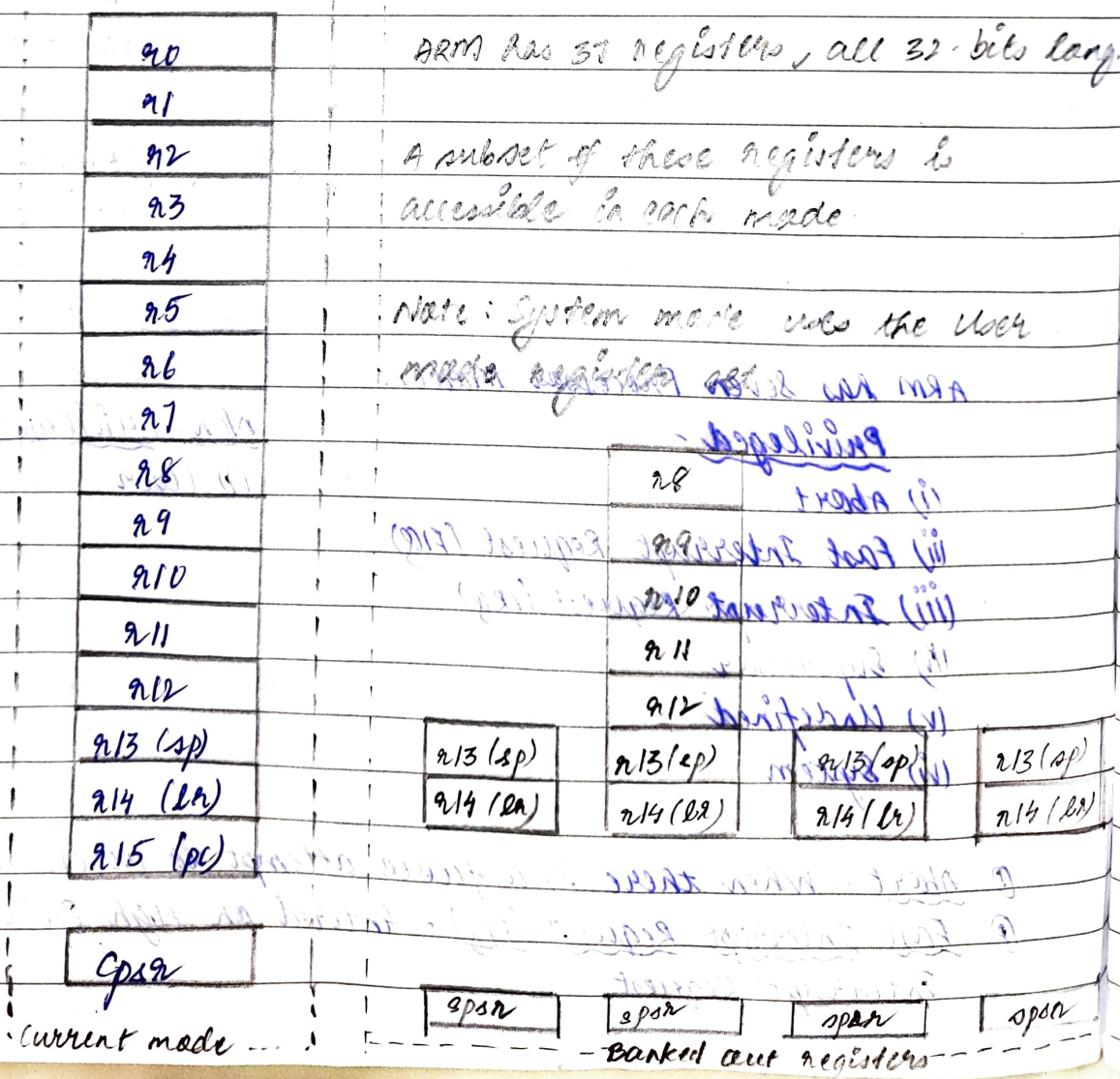
Non-Privileged -
(i) User

Data Abort - When there is a failed attempt to access memory
Fast Interrupt Request (FIQ) - Entered on High Priority Interrupt Request

(3) Interrupt Request (IRQ) - Entered on Normal Interrupt Request
(4) Supervisor mode - Mode after Reset and mode in which OS kernel executes
(5) Undefined - When the processor encounters an Undefined Instruction
(6) User - Mode in which most Applications run
(7) System - Special version of User mode that allows Full Read-Write access of CPSR.

* ARM REGISTERS:

Banked
Hidden, only visible when processor moves to corresponding mode of operation



ARM has 37 registers, all 32-bits long.

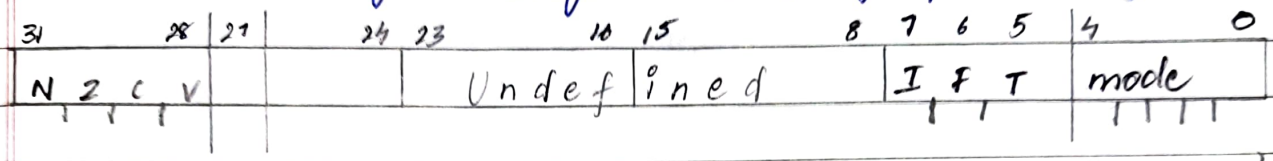
A subset of these registers is accessible in each mode

Note: System mode uses the User mode registers

SPSR registers are used to save CPSR of previous mode

* MODE CHANGING:

There are two ways to change the mode of operations -



1. Direct write to mode Bits
2. On Exception or Interrupt

m[4:0]	mode
0b10000	User
0b10001	FIQ
0b10010	IRQ
0b10011	Supervisor
0b10111	Abort
0b11011	Undefined
0b11111	System

Exceptions

SVC — Reset, FIQ, IRQ, SWI, Data Abort
Pre-fetch Abort and Undefined

Actions by processor when exceptions occur -

1. saves CPSR to SPSR of exception mode
2. saves PC to LR (R15) of exception mode — (Linked Register - store the return address)
3. sets CPSR mode Bits to corresponding exception
4. sets PC to address of exception handler

[PTO for diagrammatic representation]

* EXCEPTION PRIORITY ORDER:

1. Reset Processor
2. Data Abort
3. FIQ
4. IRQ
5. Pre-fetch Abort
6. SWI, Undefined Instruction
   - Software Interrupt

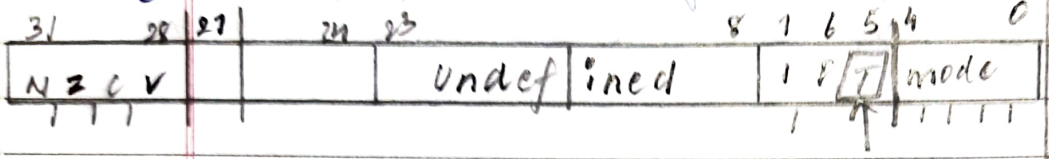
User mode

* **PROCESSOR STATES :**

Determines which instruction set is being executed.

① ARM state - 32 bit instructions

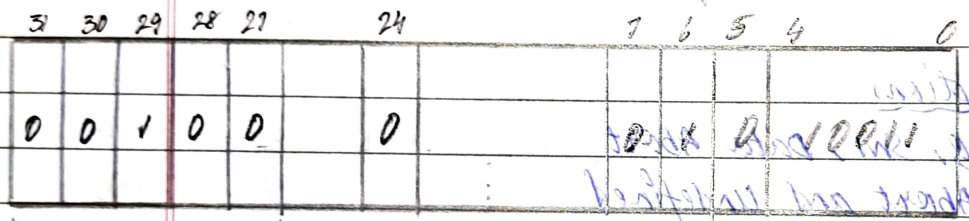
② Thumb state - 16 bit instructions



T=0 - ARM

T=1 - Thumb

* **EXERCISE - CPSR :**



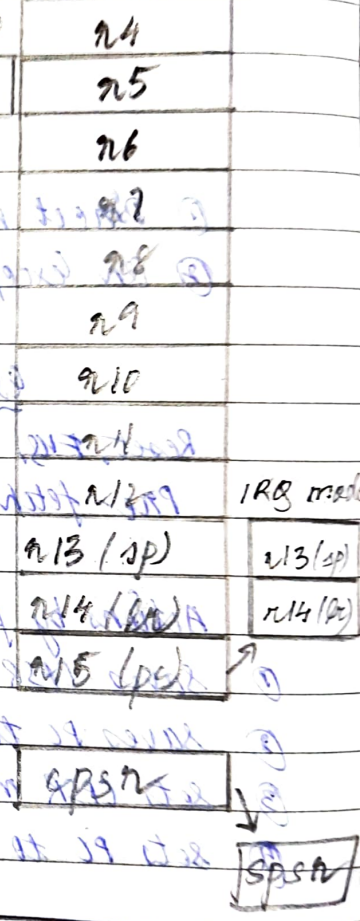
N Z C V

I - IRQ

F - FIQ

mode - Supervisory

state - ARM



LECTURE 5 - ARM INTRODUCTION SET (INTRODUCTION)

* **INSTRUCTION SETS:**

① ARM

- Standard 32-bit instruction set

② THUMB

- 16-bit compressed form of instruction set
- Code density better than most CISC
- Performance degraded

* FEATURES OF ARM INSTRUCTION SET:

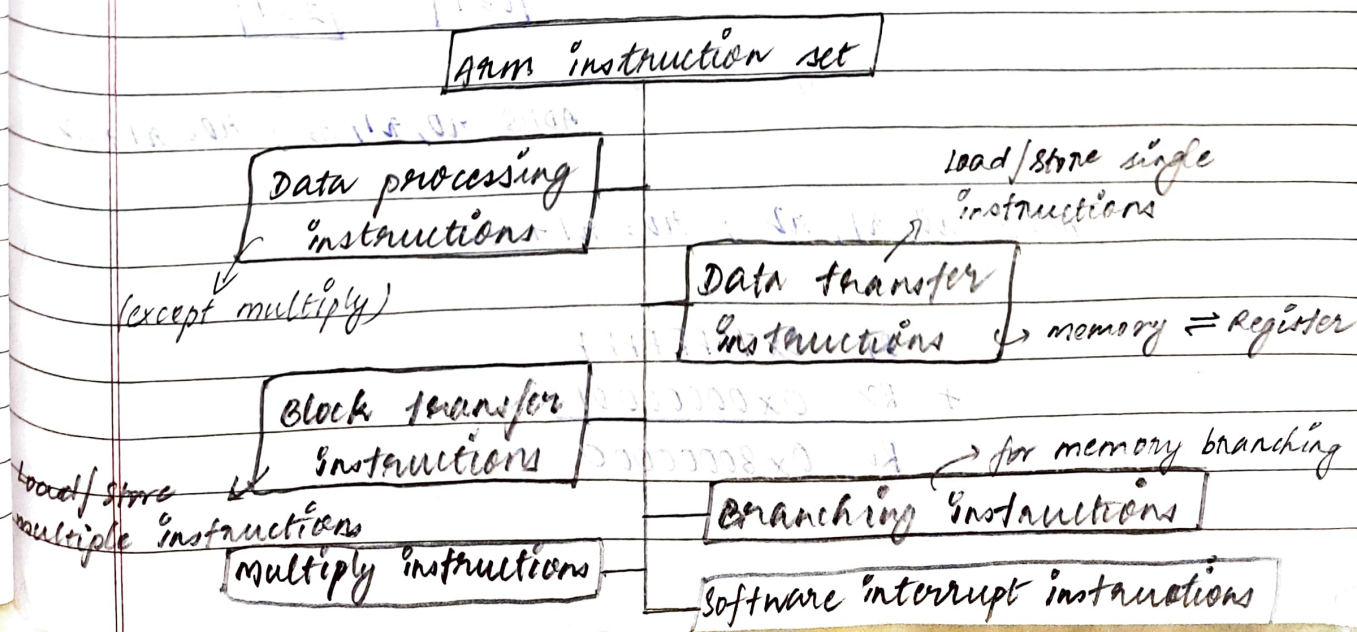
1. Load/store architecture - Memory ⇆ Register
2. 3-address data processing instructions (RESULT, FIRST OPERAND, SECOND OPERAND).
3. Conditional execution
4. Load/store multiple registers in a single instruction
5. Shift and ALU operation in single clock cycle

* CONDITIONAL EXECUTION:

- Each data processing instruction prefixed by condition code
- 16 condition codes

ADDEQ r0, r1, r2 ; if Z=1, r0 = r1 + r2

EQ	equal	MI	negative	HI	unsigned higher	GT	signed greater than
NE	not equal	PL	positive or zero	LS	unsigned lower or same	LE	signed less than or equal
CS	unsigned higher or same	VS	overflow	GE	signed greater than or equal	AL	always
CC	unsigned lower	VC	no overflow	LT	signed less than	NV	special purpose



LECTURE 6 - PART 1. DATA PROCESSING INSTRUCTIONS

* FEATURES :

1. Arithmetic and logical operations
2. 3-address format -
   (i) Two 32-bit source operands (op1 is always register, op2 is register or shifted register or immediate)
   (ii) 32-bit result placed in a register
3. Barrel shifter for op2 allows full 32-bit shift within instruction cycle. (multiple shifts in single instruction)

* ARITHMETIC OPERATIONS :

destination ← source operands / registers

```
ADD R0, R1, R2          ; R0 = R1 + R2
ADD R0, R1, #2          ; R0 = R1 + 2     (#2 : immediate data)
ADD R0, R1, R1, LSL #1  ; R0 = R1 + R1 << 1   (LSL : logic shift left)
```

```
ADD R0, R1, R2 ; R0 = R1 + R2
```

R1	0x00000001		R1	0xFFFFFFFF
+ R2	0x00000002		+ R2	0x00000001
R0	0x00000003		R0	0x00000000

C = 1 Z = 1

Status Flag Updation

```
ADDS R0, R1, R2 ; R0 = R1 + R2
```

```
ADDS R0, R1, R2 ; R0 = R1 + R2
```

	R1	0x7FFFFFFF
+	R2	0x00000001
	R0	0x80000000

MSB - number is positive

R1	0111	1111	1111	1111	1111	1111	1111	1111
+ R2	0000	0000	0000	0000	0000	0000	0000	0000
R0	1000	0000	0000	0000	0000	0000	0000	0000

MSB - number is negative

V = 1

N = 1

Hexadecimal

ADD R0, R1, #2 ; R0 = R1 + 2

R1	0x00000001
+	2
R0	0x00000003

ADD R0, R1, #1F ; R0 = R1 + 0x0000000F

R1	0x00000001
+	0x0000000F
R0	0x00000010

ADD R0, R1, R1, LSL #1 ;
R0 = R1 + R1 << 1

R1	0x00000001
+R1, LSL #1	0x00000002
R0	0x00000003

R0 = R1 + R1 << 1
= R1 + 2 x R1
= 3 x R1

→ Add with carry

ADC R0, R1, R2 ;
R0 = R1 + R2 + C

R1	0x00000001
+ R2	0x00000002
+ C	1
R0	0x00000004

SUB R0, R1, R2 ; R0 = R1 - R2

SUB R0, R2, R1 ; R0 = R2 - R1
RSB R0, R1, R2 ; R0 = R2 - R1
} - Same

Reverse Subtract

R1	0x00000003
- R2	0x00000002
R0	0x00000001

R2	0x00000002
- R1	0x00000003
R0	0xFFFFFFFF

RSBS R0, R1, R2 ; R0 = R2 - R1

R2	0x00000002
- R1	0x00000003
R0	0xFFFFFFFF

N = 1 C = 0

0xFFFFFFFF
↓ 1's Compliment
0x00000000
↓ +1
0x00000001

Addition	
C=0	No Carry
C=1	Carry

Subtraction	
C=0	Borrow
C=1	No Borrow

$$B = \sim C$$

Subtract with carry

SBC R0, R1, R2 ; R0 = R1 - R2 - B

SBC R0, R1, R2 ; R0 = R1 - R2 - ~C

	R1	0x00000003
-	R2	0x00000001
	C	0
-	~C	1
	R0	0x00000001

Reverse - Subtract with carry

RSC R0, R1, R2 ; R0 = R2 - R1 - ~C

	R2	0x00000003
-	R1	0x00000001
	C	0
-	~C	1
	R0	0x00000001

LECTURE 7 - ARM INSTRUCTION SET - DATA PROCESSING - II

* DATA PROCESSING INSTRUCTIONS :-

BIT-WISE LOGICAL OPERATIONS :

```
AND R0, R1, R2 ; R0 = R1 and R2
ORR R0, R1, R2 ; R0 = R1 or R2
EOR R0, R1, R2 ; R0 = R1 xor R2
```

R1	0x FFFFFFFF
& R2	0x00000000
R0	0x00000000

R1	1111	1111	1111	1111	1111	1111	1111	1111
& R2	0000	0000	0000	0000	0000	0000	0000	0000
R0	0000	0000	0000	0000	0000	0000	0000	0000

↗ clear a bit in a register

BIC R0, R1, R2 ; R0 = R1 and not R2

R1	1111	1111	1111	1111	1111	1111	1111	1010
R2	0000	0000	0000	0000	0000	0000	0000	0101
& Not R2	1111	1111	1111	1111	1111	1111	1111	1010
R0	1111	1111	1111	1111	1111	1111	1111	1010

R1	0xFFFFFFFF
R2	0x00000005
R0	0xFFFFFFFA

+ REGISTER MOVEMENT OPERATIONS:

↗ content of R2 will be moved to R0

MOV R0, R2 ; R0 = R2

R2	0x00000002
R0	0x00000002

R2	0000	0000	x5	0010
R0	0000	0000	x5	0010

↗ complement of R2 will be moved to R0

MVN R0, R2 ; R0 = Not R2

R2	0x00000002
R0	0xFFFFFFFD

R2	0000	0000	x5	0010
R0	1111	1111	x5	1101

PTO

* COMPARISON OPERATIONS :

→ Compares two parameters, no result stored

CMP R1, R2 ; Set CC on R1 - R2

R1 0x00000002	R1 0000 0000 ... 0010
- R2 0x00000002	- R2 0000 0000 ... 0010
Z = 1 (No Result)	Z = 1 (No Result)

R1 0x00000002	R1 0000 0000 ... 0010
- R2 0x00000003	- R2 0000 0000 ... 0011
N = 1 (No Result)	N = 1 (No Result)

CC (Condition Code field of CPSR) - N, Z, C, V

No Result

CMN R1, R2 ; Set CC on R1 + R2

R1 0x00000000	R1 0000 0000 0000
+ R2 0x00000000	+ R2 0000 0000 0000
Z = 1	Z = 1

R1 0xFFFFFFFF	R1 1111 1111 1111 1111
+ R2 0x00000001	+ R2 0000 0000 0000 0001
C = 1	C = 1

CC (Condition Code field of CPSR) - N, Z, C, V

No Result

P.T.O.

TST r1, r2 ; Set CC on r1 & r2 - Test single bit is 1 or 0 (Particular)

R1 0xFFFFFFFF	R1 1111 1111 1111 .... x4 1111
& R2 0x00000001	& R2 0000 0000 0000 .... x4 0001
Z=0	Z=0 - So the bit is '1'

R1 0xFFFFFFFE	R1 1111 1111 1111 .... x4 1110
& R2 0x00000001	& R2 0000 0000 0000 .... x4 0001
Z=1	Z=1 - So the bit is '0'

CC (Condition Code field of CPSR) - N, Z, C, V
No Result

TEQ r1, r2 ; Set CC on r1 ^ r2 - Test both numbers are equal (XOR)

R1 0xFFFFFFFE	R1 1111 1111 1111 .... x4 1110
^R2 0xFFFFFFFE	^R2 1111 1111 1111 .... x4 1110
Z=1	Z=1 Both Nos are Same

R1 0xFFFFFFFE	R1 1111 1111 1111 .... x4 1110
^R2 0xFFFFFFFF	^R2 1111 1111 1111 .... x4 1111
Z=0	

CC (Condition Code field of CPSR) - N, Z, C, V
No Result

* SHIFTED REGISTER OPERANDS :

① LSL - logical shift left by 0 to 31 places ; fill the vacated bits at the least significant end of the word with zeros.

② LSR - logical shift right by 0 to 32 places ; fill the vacated bits at the most significant end of the word with zeros.

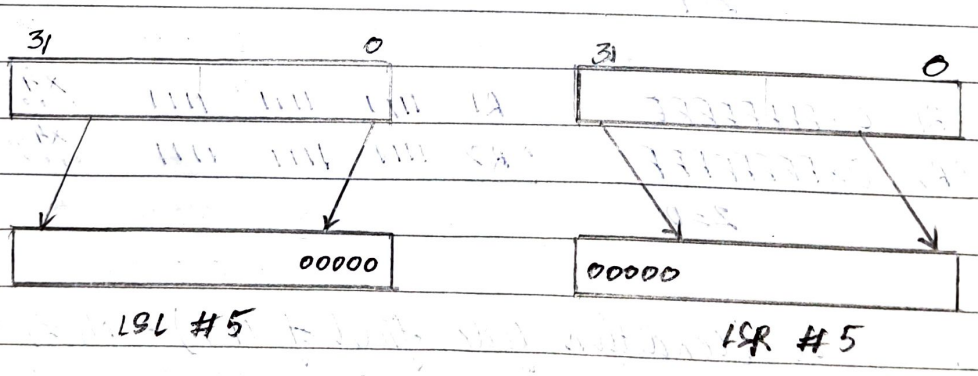
PTO

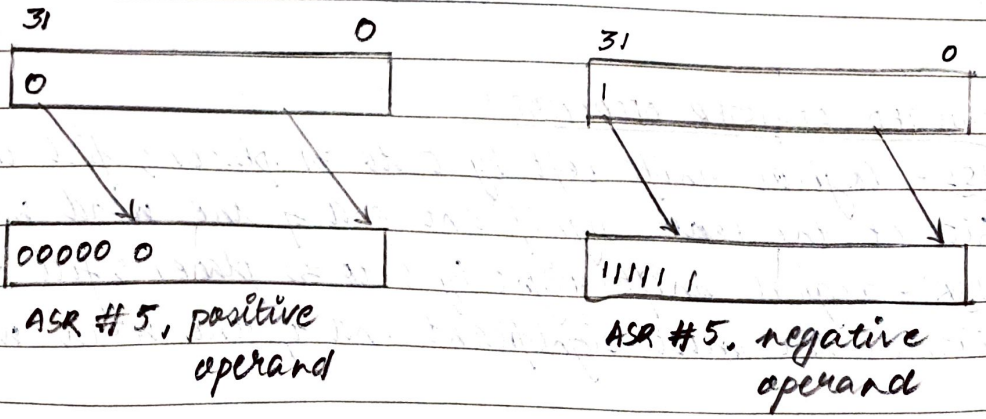
③ ASR - arithmetic shift right by 0 to 32 places; fill the vacated bits at the most important end of the word with zeros if the source operand was positive, or with ones if the source operand was negative.

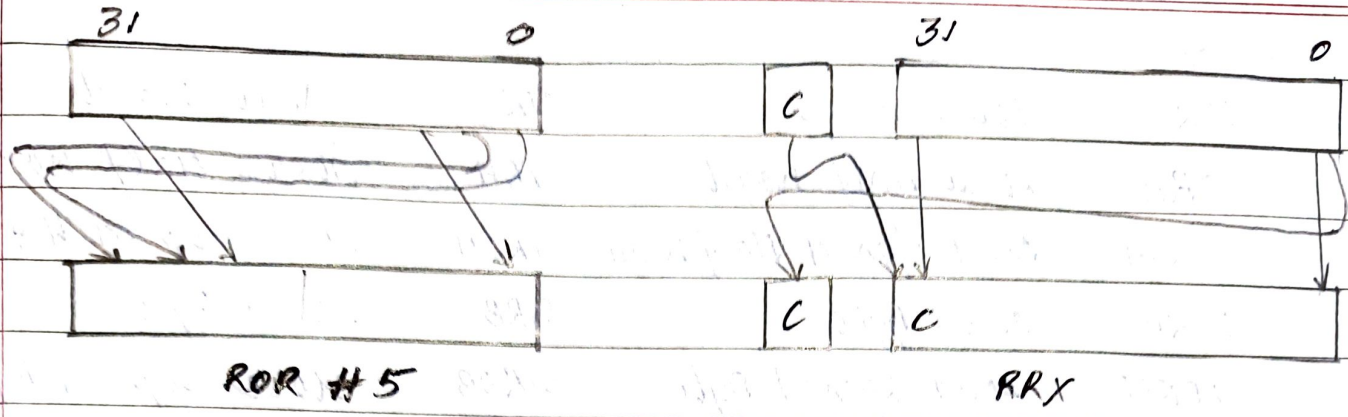
④ ROR - rotate right by 0 to 32 places; the bits which fall off the least significant end of the word are used, in order, to fill the vacated bits at the most significant end of the word.

⑤ RRX - rotate right extended by 1 place; the vacated bit (bit 31) with the old value of the C flag and the operand is shifted one place to the right. With appropriate use of the condition codes (see below) a 33-bit rotate of the operand and the C flag is performed.

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







ADD r0, r1, r1, LSL #1 ; r0 = r1 + r1 << 1

R1	0x00000001		R1	0000 ... 0001
+ R1, LSL #1	0x00000002		+ R1, LSL #1	0000 ... 0010
R0	0x00000003		R0	0000 ... 0011

r0 = r1 + r1 << 1 = r1 + 2 x r1 = 3 x r1

e.g. :

if (z == 1) R1 = R2 + (R3 + 4)

complies to

ADDEQS R1, R2, R3, LSL #2

(SINGLE INSTRUCTION !)