KAMMAVARI SANGHAM (R) - 1952

K S GROUP OF INSTITUTIONS

K.S. INSTITUTE OF TECHNOLOGY

Accredited by NAAC

(Approved by AICTE & Affiliated to VTU)

#14, Raghuvanahalli, Kanakapura Road, Bengaluru - 560109

Department of Computer Science & Design
K.S. Institute of Technology
Bengaluru - 560 109.

BLUE BOOK

Name of the student: ANAGHA SHASTRY

Sem / Section: IV - A **Branch:** CS&D

USN: 1 K S 2 1 C G 0 0 4

SUBJECT: Microcontroller & Embedded System **SUBJECT CODE :** 21CS43

MAXIMUM MARKS

Test	I	II	III	Average Marks Obtained
Date	27/06/2023	01/08/2023	04/08/2023	Test 20
Marks Obtained	20	19½=20	20	Assignment 10
Signature of Student				Activity 20
Initials of Faculty				Total Th+Pr 20+20=50
		Final Assessment		50

NAME OF FACULTY : Sweeka. Byakod

SIGNATURE OF FACULTY

Deepa

SIGNATURE OF H.O.D.

K.S. INSTITUTE OF TECHNOLOGY

First Internal Test

Q.No	Marks		Q.No	Marks	CO	CO	Total
1 (a)			2 (a)	4	CO_1		
1 (b)		OR	2 (b)	4	CO_1	CO_1	8
1 (c)			2 (c)	4	CO_1		
3 (a)	4		4 (a)		CO_2	CO_2	8
3 (b)	4	OR	4 (b)		CO_2	CO_2	4
						Grand Total	20

Second Internal Test

Q.No	Marks		Q.No	Marks	CO	CO	Total
1 (a)	4		2 (a)		CO_3		
1 (b)	4	OR	2 (b)		CO_3	CO_3	12
1 (c)	4		2 (c)		CO_3		
3 (a)			4 (a)	3 1/2	CO_2	CO_2	3 1/2
3 (b)		OR	4 (b)	4	CO_4	CO_4	4
						Grand Total	19 1/2 = 20

Third Internal Test

Q.No	Marks		Q.No	Marks	CO	CO	Total
1 (a)	4		2 (a)		CO_5		
1 (b)	4	OR	2 (b)		CO_5	CO_5	12
1 (c)	4		2 (c)		CO_5		
3 (a)			4 (a)	4	CO_4	CO_4	4
3 (b)		OR	4 (b)	4	CO_4	CO_4	4
						Grand Total	20

	A1	A2	Total
Assignment	10	10	10

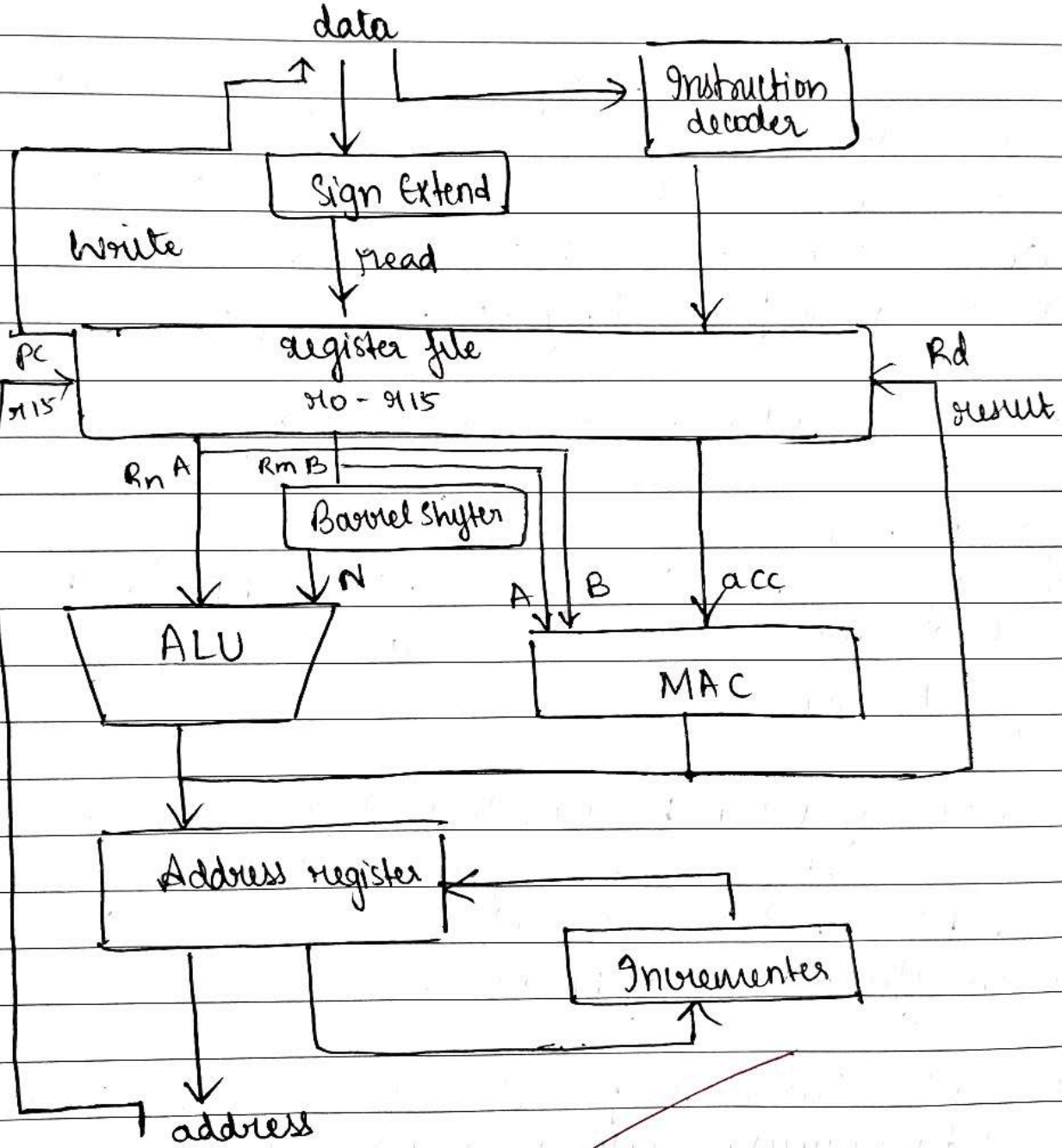
	CIE	IA		Total
Activity	15	5	20	20

SIGNATURE OF FACULTY.

1st Internals

PART-A

(a) Arm core data flow model



- arm core data flow model consists of
  lines - representing the buses
  arrows - representing the data flow
  boxes - represents either operations or storage units

- The data enters ARM core by the data bus.

- Von Neuman architecture — seperate buses for instructions & data
  Haarvard - single bus
- The size of data processed by the register file is 32 bits
- The data enters into sign extend which converts any data into 32 bits so it can be processed by the register file.

- The instruction decoder decodes instructions ~~by~~ provided.

- The register file consists of r0 - r15.
  Here the register file gives input to ALU & MAC (arithemetic logical unit & multiplier accumulator unit) which performs the required action.

- The barrer shifter is a hardware tool which shifts the data to the left or right by ~~using~~ using LSL, ~~RSL~~ LSR etc before being processed according to the instruction.
  This is an effivient tool of ARM core.

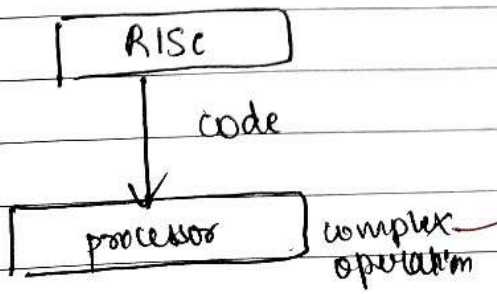
- Source registers Rm & Rn provide source operands to MAC which performs various operations as required

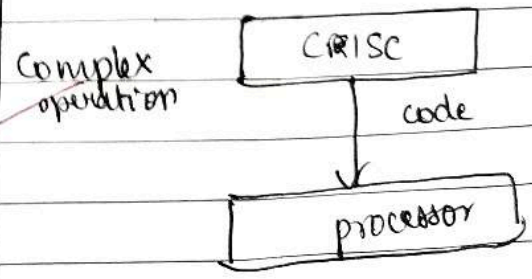
- The result from ALU & MAC are stored into destination register Rd

- The address of the result is stored in the address register which then goes to incrementer. This points at the next instruction required from register file

2(b)

RISC	CISC
• Reduced instruction set computer	• Complex instruction set computer
• reduced instructions thus less the number classes instructions can be formed by using simpler instruction	• complex instructions thus it has to be written & implemented as it is
• the instruction is of variable length	• the instruction is of fixed length
• uses pipelining	• no pipelining
• as it uses pipelining, no micro-code required	• uses microcode to execute instruction
• There are many general purpose registers which store any data or address	• only specific general purpose registers with specified instructions
• load store instruction has seperate accessing & data processing	• same instruction to access & process data
• simpler	• complex
• flexible	• not flexible
• accurate, efficient	• less efficient
• complexity less	• complexity more





4 major rules of RISC design:

① Instructions
The RISC processor uses reduced instruction classes.
Thus programmer / compiler can use simpler instructions & combine them to get complex instruction required.
The instructions are of variable length

② Pipelining
Instructions are broken down into many sub units so as to have parallel processing & reduce time

③ Registers
RISC uses many general purpose registers. Thus the data & address can be stored in any GPR

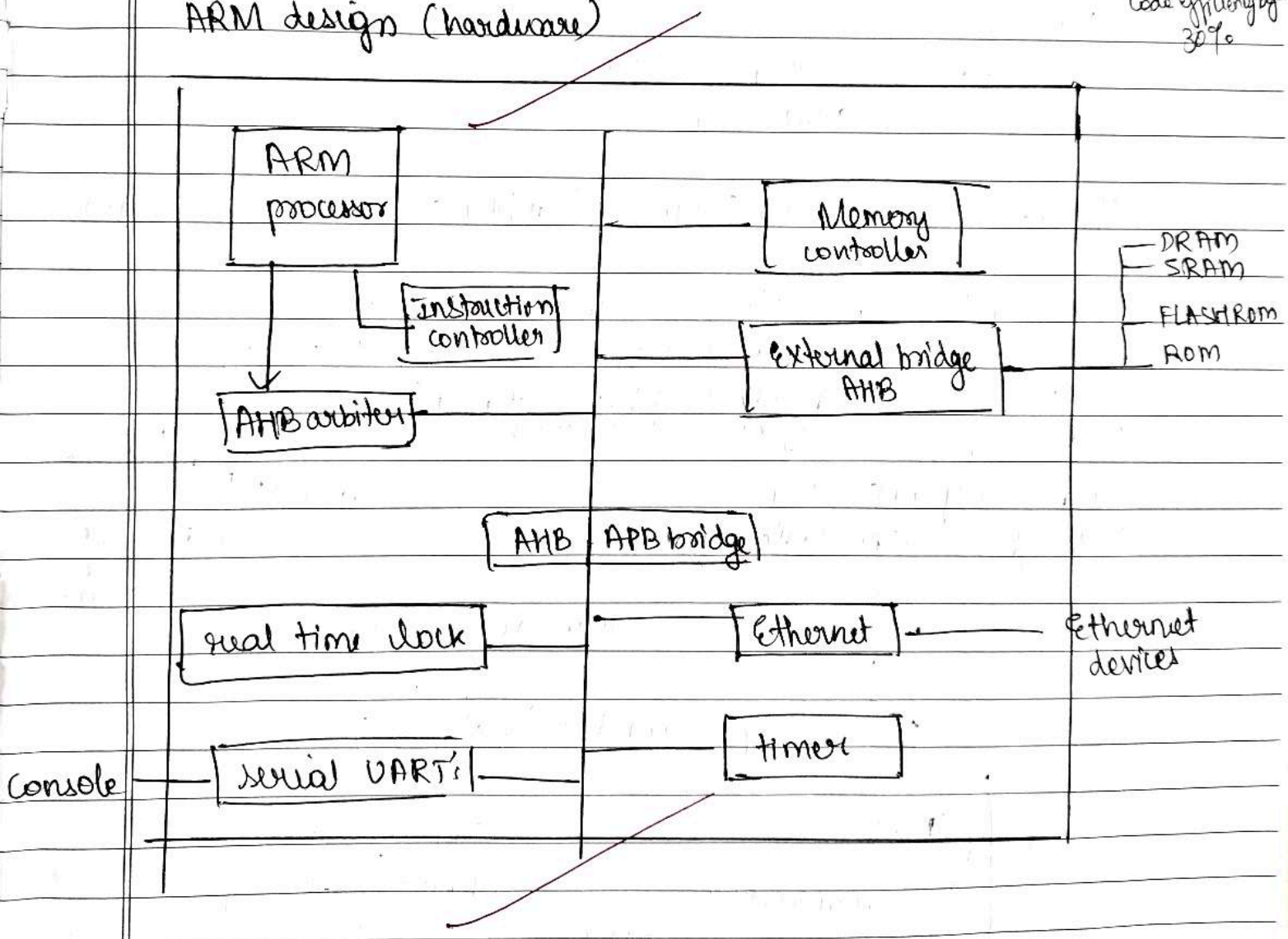
④ Load & store instruction
This includes seperate load (LDR) & store (STR) for data processing.
Here the data is processed & registers accessed seperately

2(c) ARM design philosophy

→ variable cycle execution
→ inline barrel shifter – shifting data before processing instruction
→ thumb arm set – used to solve extra set of 32 bit instruction
→ conditional branches – increase core performance & code density
→ Effiient instructions – uses DSP to solve 16x16 multiplier instructions

Increases code efficiency by 30%

ARM design (hardware)



ARM philosophy:
includes 4 units:

1. armprocessor - controls embedded device
   has core & other surrounding device (MMU, cache)
2. controller - coordinates functional units
3. peripherals - i/p & o/p operation external/on chip
4. bus - communication link between all devices in the hardware

AMBA - Advanced microcontroller Bus Architecture developed in 1996
has:

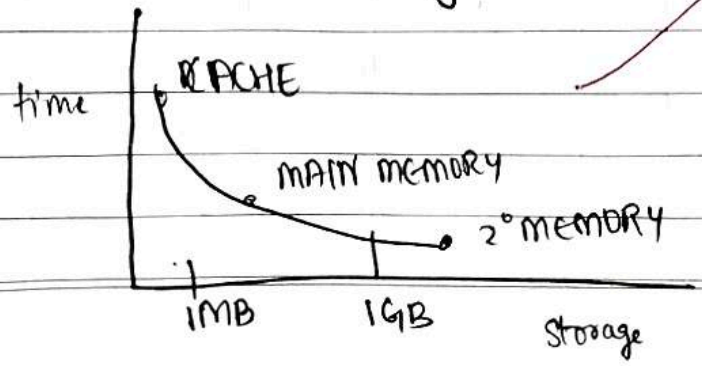
1. APB - arm processor bus
2. ASB - arm system bus

The arm processor bus & arm system bus were initial designs

Further AHB: arm high performance bus was used
→ provided higher data throughput & complex multiplexed
→ bus structure which was better than ASB
→ It also provides higher speed 64/128 bits clock cycles

Types of AHB:
- multilayer AHB (many bus masters)
- AHB Lite (only one bus master)

The Memory hierarchy:
taste off - cache faster & small
↓
main memory slower than cache & larger
↓
2° memory (secondary) (slowest & largest)



types of memory

DRAM - dynamic random access memory
(uses refreshing & stores data in form of charge)

SDRAM - Synchronous DRAM
uses pipelining & burst cycle

SRAM - synchronous DRAM
(faster than DRAM)

controllers — memory controller
interrupt controller — simple IC, vector IC

memory controller controls the memory connected to bus

Interrupt controller - controls the interrupts

Standard uses interrupt masks (I & F)

Vector Uses general to provide interrupt priority (provides interrupt to device with greater priority)

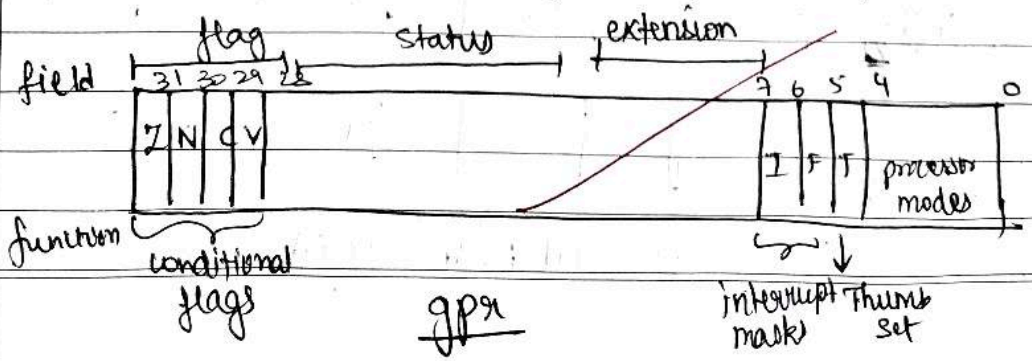
Peripherals - external communicator of embedded system
↳ provides the input output connecting to external chip

The structure also contains ethernet devices, & UART connected to it

PART-B

3(a) CPSR

current program status register
consists of 4 fields of 8 bit each
- Flag - has conditional flags
- Status
- Extension
- Consists of programming modes & Interrupt masks



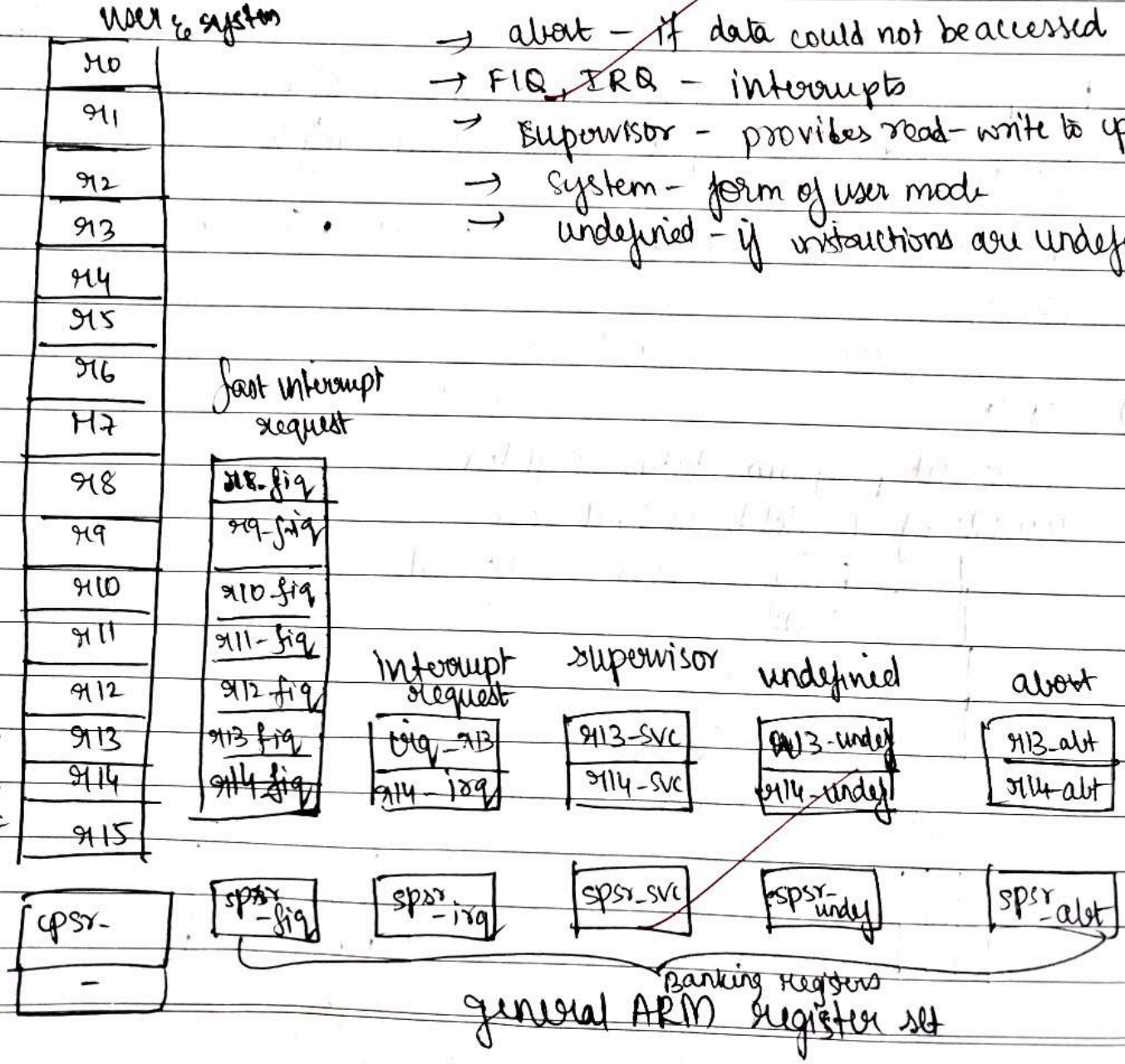
gpr

Processing modes

- privileged (6) → rest
- non privileged (1) → only user mode

These include
abort (abt), fast ~~interrupt~~ request (~~IRQ~~ FIQ), interrupt request (irq), Supervisor (svc), System (sys), user (user), undefined (undef)

- → abort – if data could not be accessed
- → FIQ, IRQ – interrupts
- → Supervisor – provides read-write to cpsr
- → System – form of user mode
- → undefined – if instructions are undefined



Banked registers:
There are 37 registers out of which 20 are banked.
These are hidden from program & once mode is called, the registers are used

Eg: r13, r14 in abort mode change to r13 - SVC, r14_SVC in supervisor mode

Conditional flags
Z - zero
N - negative
V - overflow
C - carry

Thumb set has T = 1
instruction set = 16 bits
core instruction = 30
has seperate barrel shifter & ALU
has 8 general purpose registers, 7 high purpose registers + pc

Interrupt masks

prevents interrupt to stop the processing using masks I & F (7 & 6)

I masks FIQ
F masks IRQ

3(b) Branch instructions

- programmers use this instead of if, else
- used to call subroutine
- here one execution of an instruction stopped to call sub-routine
- used for 'if-else-for' statements

SYNTAX: B {B} <cond> label
BL {B} <cond> label
BLX {B} <cond> label/Rm

B	branch	pc - label lr - next instruction
BL	branch with link	pc - 0xfffffffe, T = 1
BLX	branch link with extension	pc - label pc = 0xfffffffe T = Rm & 1 lr - next instruction

Here these instructions are used with labels
Labels are pc related offsets which are store either forward or backward branching

Branch instructions labels - 32 or 64 bit store labels

Branch instructions can be used to handle vector interrupts & exceptions

Eg code:

```
B   forward
ADD x10,x11,#1
ADD x12,x13,#4
SUB x16,x17,#5
```

Here branch forward causes forwarding before the instructions are executed.

```
ADD x10,x11,#1
ADD x12,x13,#4
SUB x16,x17,#5
B   backward
```

Here branching occurs after instructions in the code

4

→ X —

2) a) 4
b) 4
c) 4
2) a) 4
b) 4
20

20
20

Good

Sri
28/6/2023

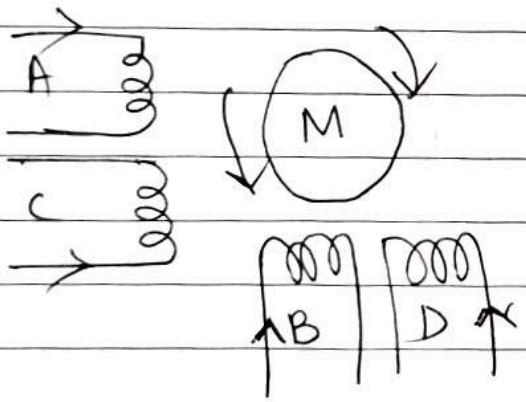
2nd INTERNALS

PART-A

11(a) • Stepper Motor is electromechanical device that produces discrete displacement on dc current.
DC Motor, produces continuous displacement on dc current.

• Two types — Bipolar
Unipolar

Unipolar - 2 windings per phase
• to change direction of rotation, the current direction is changed.
• coil A & C have current in opposite directions in one phase.
coil B & D have current flowing in opposite direction in the second phase



Bipolar - 1 winding per phase
• to change direction of rotation, dynamically the current is changed
• the current changes direction of rotation

Stepping up of stepper motor:

① Full step: two coils ~~are~~ / phases energised simultaneously

Steps	coil A	B	C	D
1	H	H	L	L
2	L	H	H	L
3	L	L	H	H
4	H	L	L	H

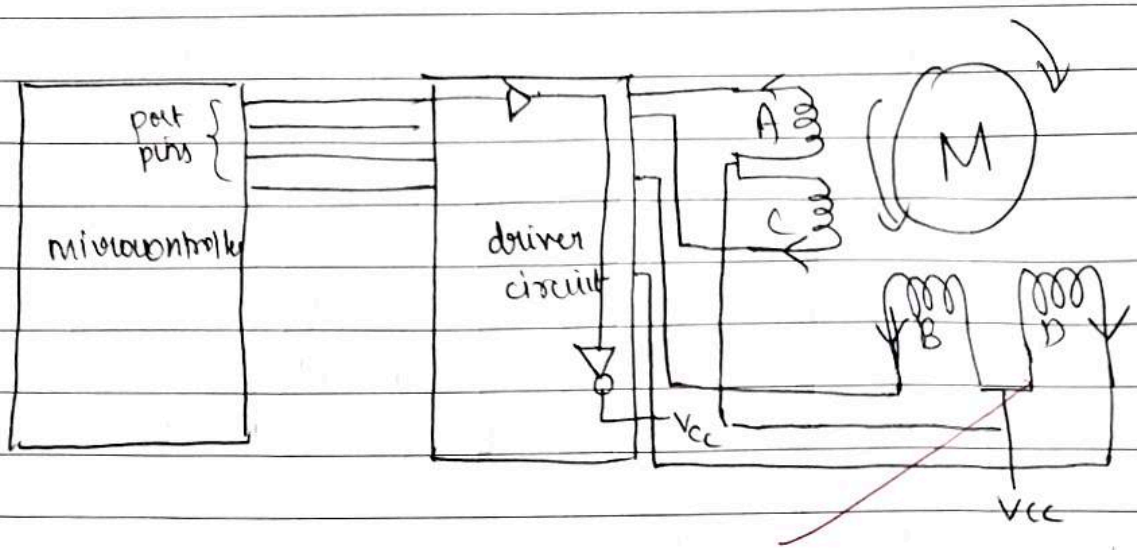
② ~~Ha~~ Wave step: one coil energised

Step	coil A	coil B	coil C	coil D
1	H	L	L	L
2	L	H	L	L
3	L	L	H	L
4	L	L	L	H

③ Half step: combines both full & wave step

Step	coil A	coil B	coil C	coil D
1	H	L	L	L
2	H	H	L	L
3	L	H	L	L
4	L	H	H	L
5	L	L	H	L
6	L	L	H	H
7	L	L	L	H
8	H	L	L	H

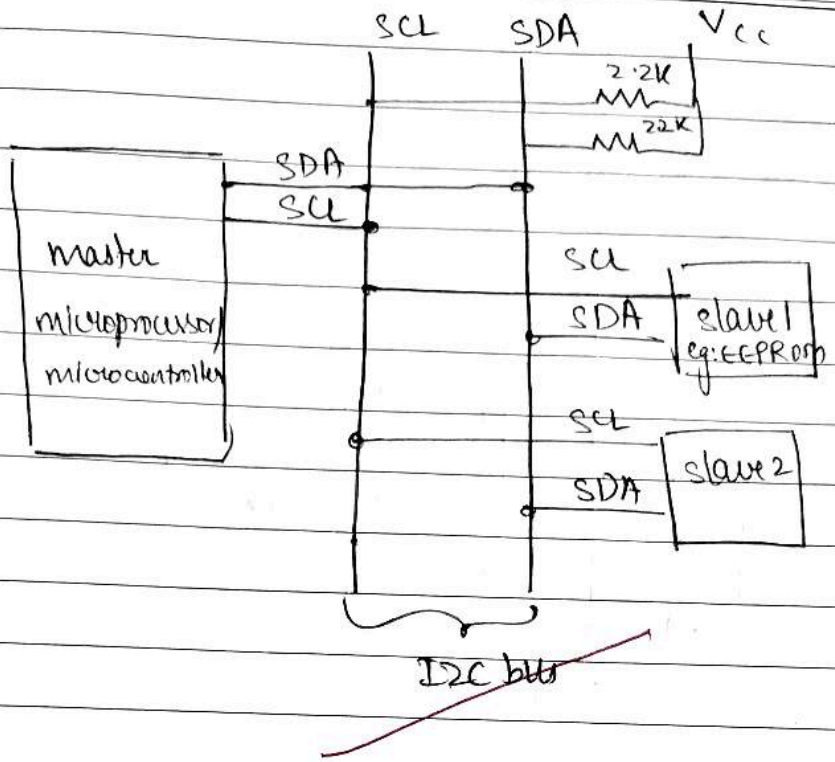
interfacing stepper motor using driver circuit



1(b) On board communication interfaces include the communication techniques between IC & peripherals of embedded system.

① I2C : Inter integrated circuit bus

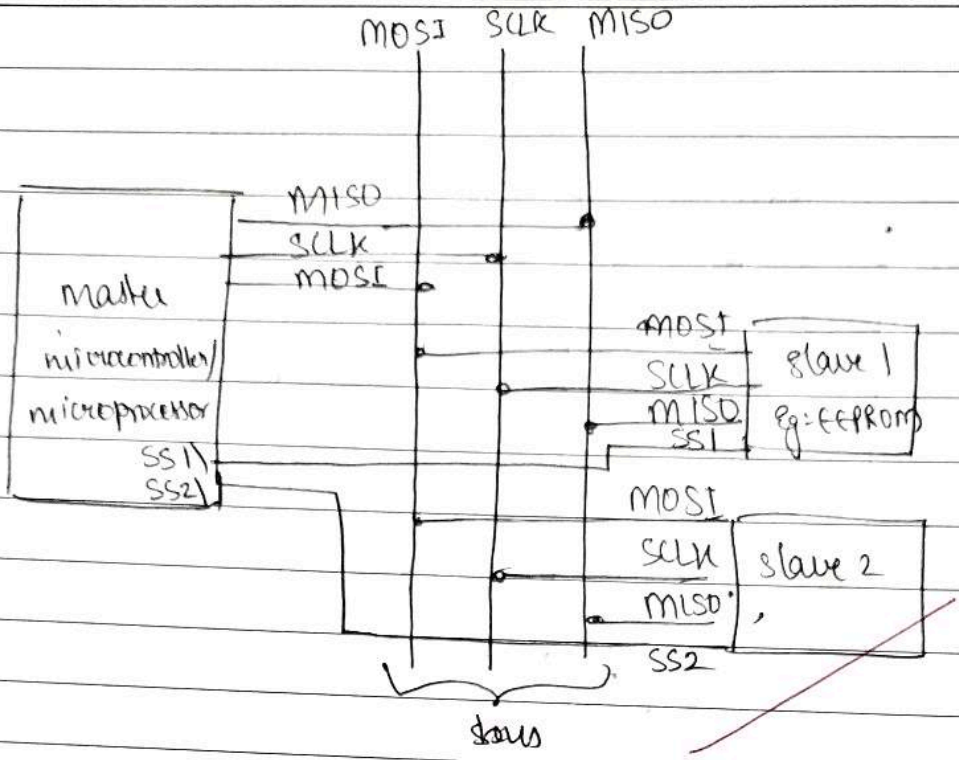
- bi synchronous, bidirectional, half duplex, 2 wired bus
- has 2 bus lines — SCL : synchronous clock to generate pulses
  SDA : synchronous data for data communication
- the devices follow master - slave relationship where master handles communication & slave waits for master
- master-slave are like transmitter - receiver
- the when bus line is idle, the open drain is is floating state & the SCL - SDA are in high impedance state



② SPL : serial peripheral communication bus

- synchronous, bidirectional, full duplex, 4 wired bus
- one master & many slaves
- 4 bus lines
  - MOSI : master out slave in
  - MISO: master in slave out
  - SCLK : serial clock
  - SS: slave select

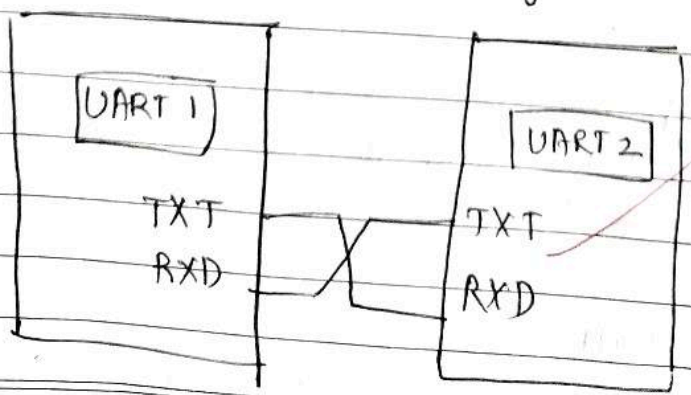
- MOSI: to communicate data from master to slave
  MISO : to communicate data from slave to master
  CLK : for clock pulses from master
  SS: selects slave device to send communication
- when SS is chosen, the slave device selected by master for communication
- it is like shift register
- The 4 bus lines are in floating high impedence state when master is idle.



③ UART: universal asynchronous receiver transmitter

- consists of 2 devices asynchronous communication
- receiver : receives data
  transmitter: sends data

- start & stop bits used to indicate start of transmission & end of transmission respectively.
- Parity bits used for error checking.



TXT : transmission
RXD : receiving

1(c) Deadlocks: it is when a process holds data(resource), and does not give up the data for other processes to use.

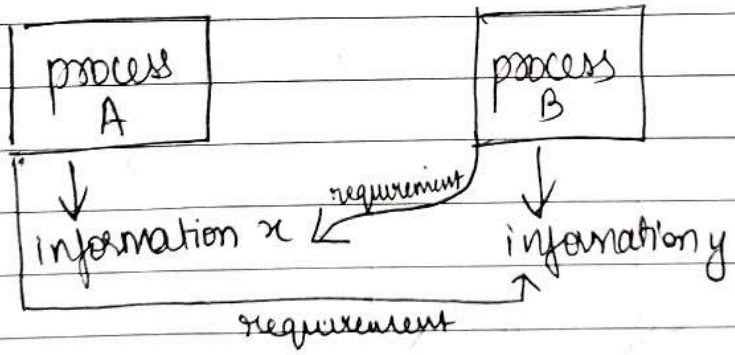
deadlocks is created when:

Eg: process A holds information x & process B holds information y. Process B wants information x held by A & process A wants information y held by y.
Process A won't leave the information x until it receives y. But since B requires x, it will not be able to execute

The deadlocks do not complete process execution as the processes are waiting to receive information from another process

Conditions:

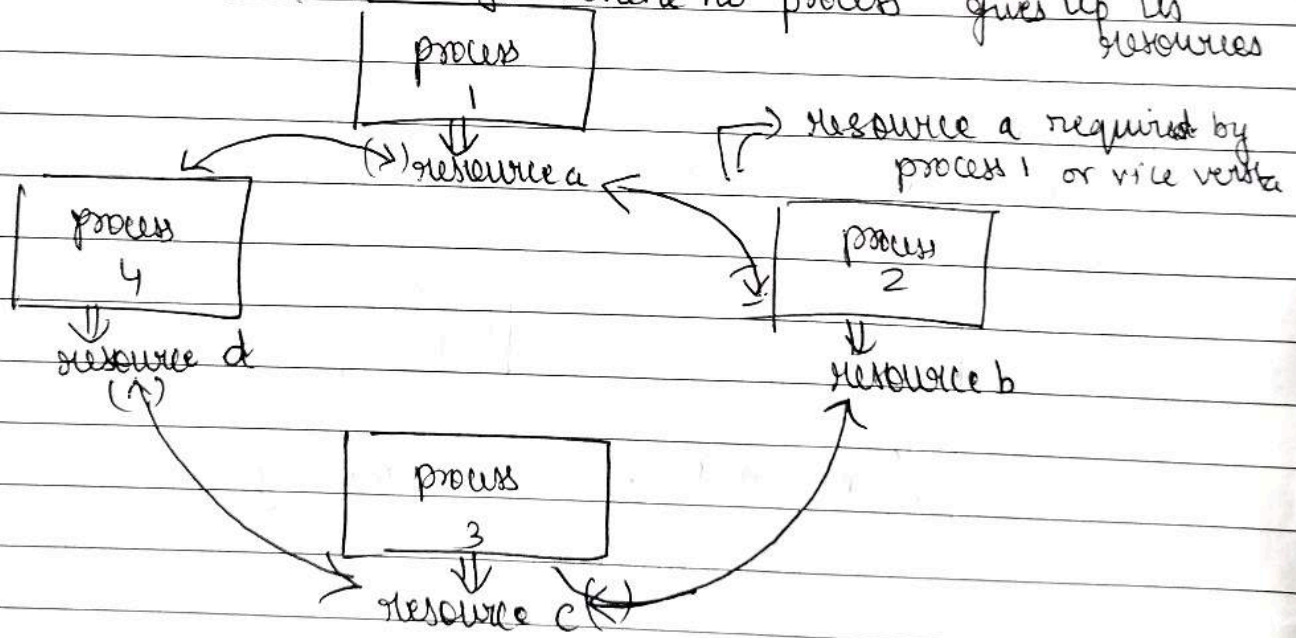
① Mutual exclusion
a process can hold only one resource at a time

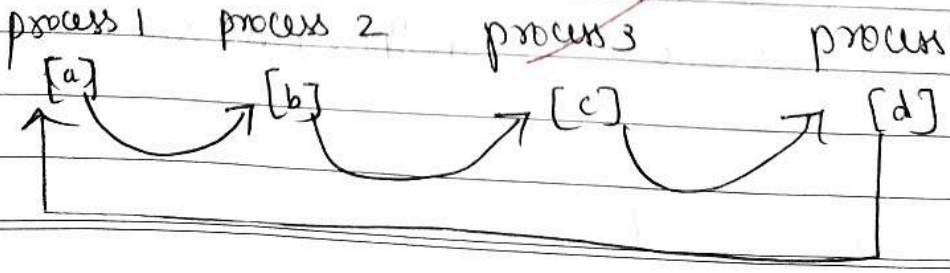


② Hold and wait
Process holds onto the resource it has and waits to receive another resource from another process.

③ No preemption -
a process will give up its resources only when it wants. This means that process will keep the ~~resource~~ resource & not let the resource be preempted / given to another process

④ Circular wait
Here a process ~~no~~ 1 holds onto resource a. & requires resource b held by process 2.
process 2 requires resource c held by process 3
process 3 requires resource d held by process 4
But process 4 requires resource a held by process 1
This causes a vicious cycle where no process gives up its resources





PART - B

4(b) • Firmware is a tool in embedded system which uploads necessary updates & initial conditions into the program.

• Firmware used to solve algorithms, interrupt handles, clock timing etc.

Embedded firmware design:

• Firmware can be loaded into the program. This can be done:

① by either writing program in high level embedded C/C++ etc

② or by writing code in ~~AA~~ easy assembly language

• The firmware in high level languages is understood & read. This is because of the IDE which contains debuggers, compilers etc the modifications in the program must be done based on the language used & with help of IDE

• If the program is written in assembly language, third party tools help convert it into codes which firmware uses.

• The codes written in either high level language or in assembly language must be converted into machine readable code. This is done by third party tools or using inbuilt system micro controllers/ microprocessors.

• To convert the codes into machine readable codes: this process is called Hex file creation i.e .hex file

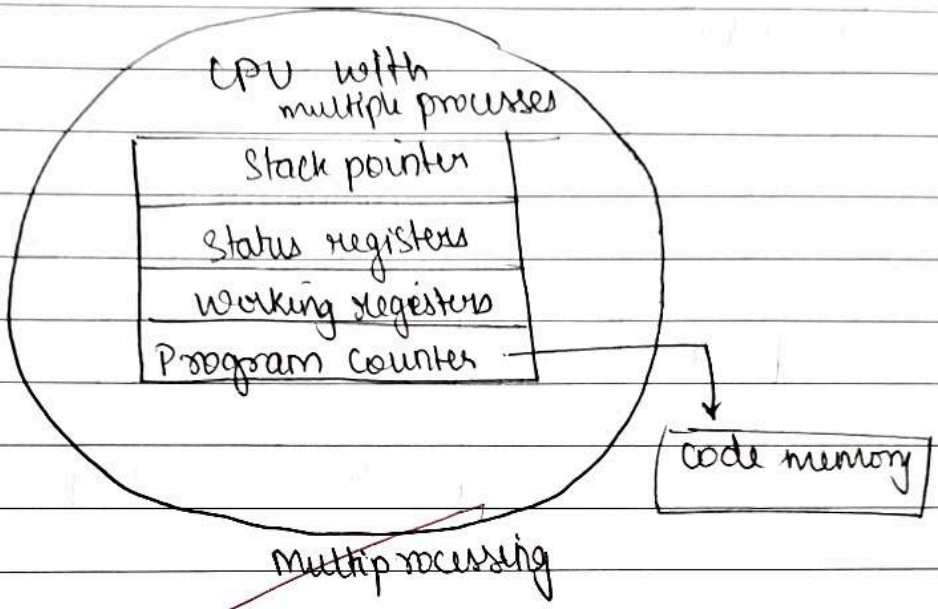
4(b)

Multiprocessing	Multitasking
• when multiple processes are running at same time	• when the OS decides to switch from one running process to another.
• maximum utilization of CPU	• maximum utilisation of OS
• CPU has shared memory for all processes	• CPU need not share memory
• CPU utilization, no context switching	• Context switch, context save & retrieve
• no interruption in execution	• one process interrupted to begin another process
• the processes execute simultaneously	• one process executes & further stops for another process to execute

Multiprocessing

- Here multiple processes are running at the same time
- This increases efficiency of the CPU
- The tasks/processes have shared memory which includes shared program memory & shared data memory

- multiprocessing allows processes to communicate with each other.
- the processes mimic processor to use all properties of cpu.



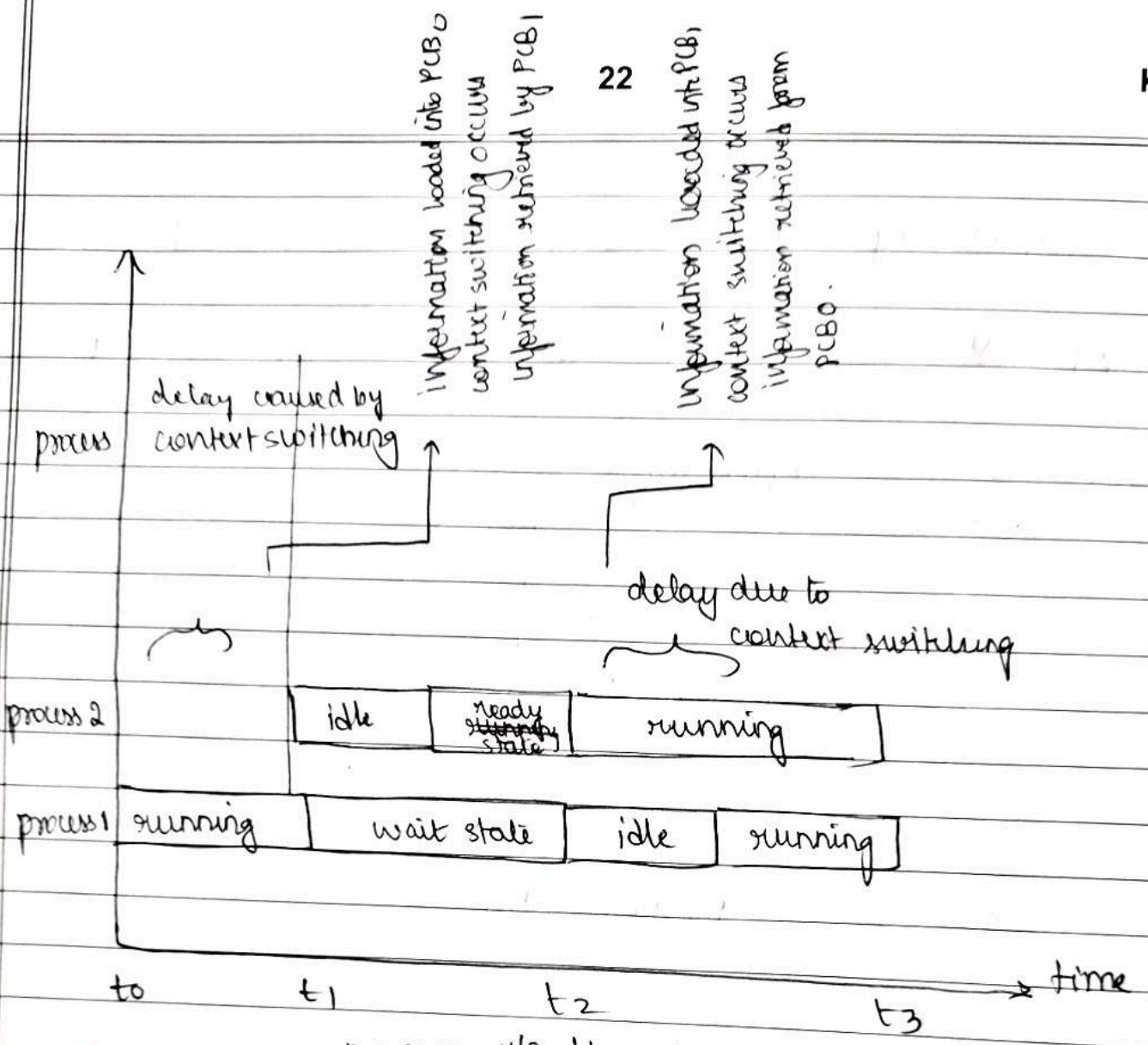
Multiprog multitasking

- involves ~~an~~ OS to stop execution of one process & begin execution of next process.
- Here it includes context - switching

Context switching involves the information of one process to be stored in a register. so that OS begins execution of next process the first process is in wait state.

Context saving: saving information so that it can be used ~~afte~~ once the process is executed

Context retrieval: to retrieve information stored from registers



process v/s time in multitasking

1) a) 4
b) 4
c) 4
4) a) 3½
b) 4

19½

20
20

INTERNALS-2

1(a) C program to print square of integers:

```
#include < stdio.h >
int square (int i);
int main (void)
{
   int i;
   for (i = 0 ; i < 10; i++)
        {
            printf ("square of the number %d is %d", i , square (i));
        }
    }
int square (int i)
 {
    return i * i;
 }
```

⇒ To convert this program to assembly function, change the declaration of the program

Use asasam assembler to generate square.s.

code:

```
AREA |.text|, CODE, READONLY
int square(int i);
square,
MUL r1, r0, r0 ; r1 = r0 * r0
MOV r0, r1     ; r0 = r1
MOV pc, lr; Return 0;
END
```

1(b) Profiling & Cycle Counting

- The first step of in any optimisation technique is to check for the assignment of cycles
- Thus the profiler is a tool which check the time taken/ processor cycle time taken during an ARM function
- The cycle counting is a process where the number of cycles being executed in the ARM instruction is checked.
- Cycle checking helps in creating a benchmark
- Cycle checking helps in checking cycles before & after the optimisation has taken place.
- The instructions being executed with conditional codes with respect to conditionals on cpsr have to use one cycle if the condition is not met

  If condition is met, it depends on the operation:

  ALU → 1 cycle - with register shift & 2 cycles with pc shift

  N-32 bit / N-16 bit load / store → N cycles

  branching → 3 cycles

  multiplication → depends on 2nd operand
- The profiling & cycle counting is done by the ARM ~~ADSI.1~~ ADSI.1 debugger called 'ARMulator'
- The ARMulator has a hit counter. ~~The~~ It is related to the program counter. Every time the pc points at a function, the hit counter is updated.
- There is no such debugger/ device for cycle counting but is is done by using ARM debugger & ARM processor/ together compiler

1(c) Unaligned Data & Endianness

Unaligned Data

- this is a major drawback for memory access & processing and thus must be avoided
- Here the bits can be aligned bit wise ~~not~~ by the compiler.
- The load, store instructions are not aligned, are stored anywhere 'byte- respectively'
- Thus since data is unaligned, memory access is not effective.
- The '__packed' is a keyword which is used to align data in the structures/memory without use of padding.

Eg:

with padding ← typedef struct{ → using __packed before typedef:

```
int/char a;
int b;
char c;
short d;
```

	+3	+2	+1	+0
+0	pad	pad	pad	a
+4	b[31,24]	b[23,16]	b[15,8]	b[7,0]
+8	d[15,8]	d[7,0]	pad	c

Here 20 cycles used

	+3	+2	+1	+0
+0	b[23,16]	b[15,8]	b[7,0]	a
+4	d[15,8]	d[7,0]	c	b[31,24]

Here 3 cycles used

even though the number of cycles are reduced the data is unaligned & similar data not together

thus __packed is better only to reduce space but speed is issue as the data is unaligned ←

- Eg The __packed caused unaligned memory access which leads to speed related issues

- The -packed can be used when data is shifted from the ARM to processor.

Here ARM supports unaligned data but processor does not. Thus causing inefficiency.

- Eg program for -packed:

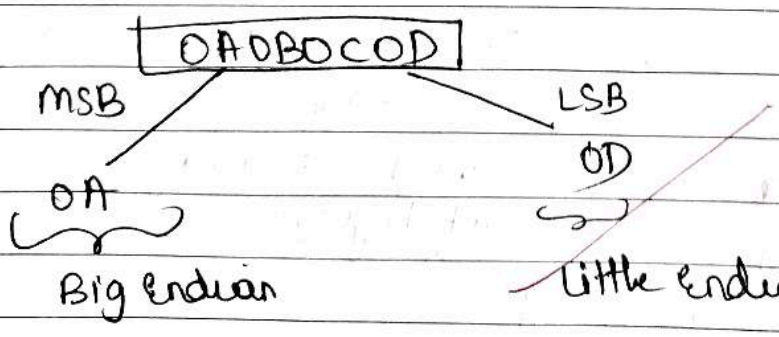
```
int read (__packed int data)
{
return data;
}
```

Here -packed can be replaced by (type char *) to prevent unaligned data.

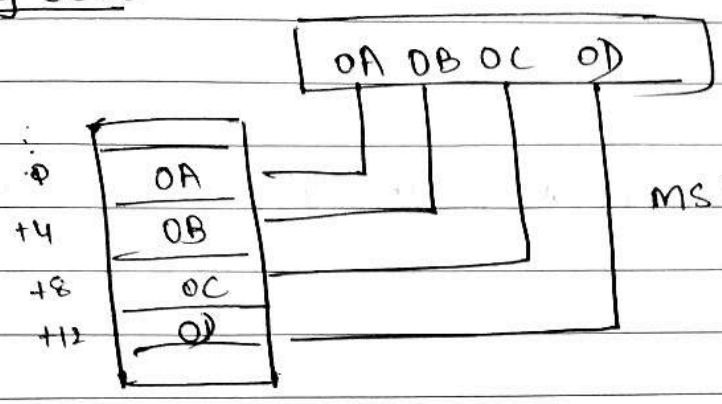
Endianness -

- it is the sequential organisation of data ~~word~~ / bytes respectively

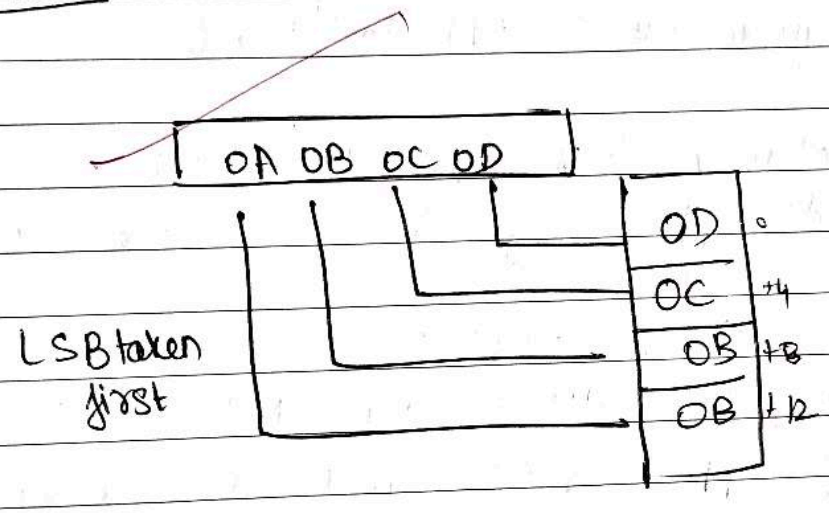
① Big Endian: ~~differentiates~~ Takes Most Significant Byte first
② Little Endian: takes least significant Byte first



Big Endian



little Endian



The Big Endian & little Endian causes the storing of bytes differently

└ this causes unaligned data

↓

this causes inefficiency

4

Thus either little / Big Endian used by entire processor.

Eg: in Structural arrangements, & bit fields little Endian used.

PART - B

4(a) To choose RTOS, we consider the functional & non functional parameters

Functional

1. processor support: all RTOS do not support all types of processor but the processor used must be supported by RTOS
2. memory requirement: ROM to store OS files, RAM for OS services
3. Real time behavior: it is important for RTOS
4. Kernel & interrupt latency: Kernels can store interrupts that cause interrupt latency
5. Task synchronisation & Inter process communication: OS dependent
6. Language support: JVM for Java, .NET for Microsoft
7. Modularity support: programmer can choose operations & recompile OS
8. Debugging & development tools needed

Non Functional

1. Off the shelf / custom made based on requirement
2. cost - effective
3. real time behavior, debugging tools
4. Efficient & ease of use
5. after sales like updates, bug fixes, onservice - calls

Integration & testing

- Here the firmware has to be embedded into the target specified hardware
- this is 'embedded intelligence'
- Once the firmware is embedded, the area in the hardware is checked
- If there is available area in code area & if the firmware fits, then, the firmware is downloaded
- If there is not enough code memory or the firmware is to big to fit, then it is integrated / interfaced by using external flash ~~of~~ chips like EPROM or Flash.
- The firmware can also be downloaded for free from the internet / or can be commerical.

Testing

① after firmware is downloaded / interfaced:
   ↳ registers are checked (argument, variable, scratch registers)
② the I/O operations checked
③ debugging & after sales checked
④ the software routines needed by hardware checked

4(b)
```
#include <LPC21xx.H>
unsigned int delay = 0, Switchcount = 0;
int main (void)
{
    PINSEL1
    IO0DIR
```

```
#include <LPC21xx.H>
unsigned long int delay = 0; Switchcount = 0;
unsigned int Disp[16] = [ 0x003F0000, 0x00060000, 0x005B0000,
0x004F0000, 0x00660000, 0x006D0000, 0x007D0000, 0x00070000,
0x007F0000, 0x006F0000, 0x00770000, 0x007C0000, 0x00390000,
0x00340000, 0x00560000, 0x00790000, 0x00710000]
#define ALLDISP 0XF0000000;
int main (void)
{
    PINSEL1 = 0X00000000;
    IO0DIR = 0XF0FF0000;
    IO0SET = ALLDISP;
    IO0CLR = 0X0F000000;
    for(delay = 0; delay < 100; delay++)
      IO0SET = ALLDISP; Disp(Switchcount);
    for(delay = 0; delay < 10,00000; delay++)
      IO0CLR Switchcount++;
      IO0CLR = 0X0F000000;
      if(Switchcount == 16)
        {
          IO0CLR = 0XOFF00000;
          Switchcount = 0;
        }
    IO0CLR = 0XF0000000;
}
```

(or)

```
int main (void)
  {
    PINSEL1 = 0x00000000;
    IO0DIR = 0xF0FF0000;
    IO0SET = ALLDISP;
    for (delay=0 ; delay < 100; delay++)
    IO0CLR = Disp (switchCount);
    for (delay=0; delay < 1000,000; delay ++)
    SwitchCount ++;
    if (SwitchCount == 16)
        {
          IO0CLR = 0x0FF00000;
          SwitchCount = 0;
        }
    IO0CLR = 0xF0000000;
  }
```

//Here : 0x003F0000 → 0
0x00060000 → 1
" 5B " → 2
" 4F " → 3
" 66 " → 4
" 6D " → 5
" 7D " → 6
" 07 " → 7
" 7F " → 8
" 6F " → 9

A ← 0x00770000
B ← 0x " 7C "
C ← 0x " 39 "
D ← 0x " 5E "
E ← 0x " 79 "
F ← 0x " 71 "

This is displayed

1) a) 4
   b) 4
   c) 4
4) a) 4
   b) 4

20/20

9/9/22