Current Program Status Register (CPSR):-

→ Stores the ~~result of the~~ status of the result of a particular operation.

→ Zero (Z) → 0

Carry (C) and Overflow (V) and Neglit (N) } Status bits available for various possible results.

Carry bit is used for unsigned data and Overflow is checked when operated on signed data. The Status Register changes its value for every instruction executed.

Data Instructions:-

Arithmetic :- ADD, ADC (add with carry), SUB, SBC (Subtract with Carry), RSB (Reverse Subtract), RSC (Reverse Subtract with Carry), MUL, MLA (multiply and accumulate

Logical Instructions:- AND, ORR, EOR (Exclusive OR), BIC (Bit Clear). ANDNOT.

Comparison:- CMP (Subtract and compare) → They change flags.

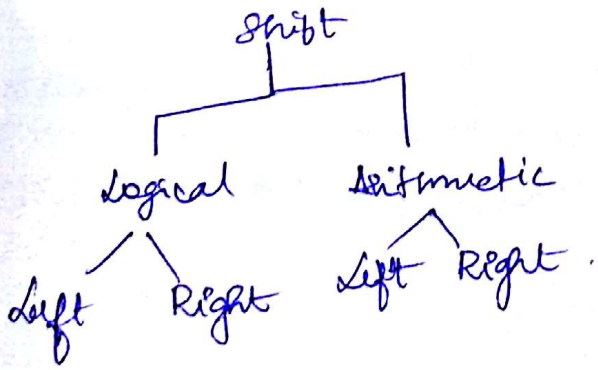
TST (Bitwise Test) → $R_1 \wedge R_2$

CMN (Negated compare). → CMN $R_1, R_2 = R_1 + R_2$.

TEQ (Bitwise negated Test) → R_1 exor R_2.

10/7/2017

Shift and Rotate Instructions.



Eg:- Shift 1101 2 times.

1101 Shifted twice to Left is

0100 → Logical & Arithmetic Left.

0011 → Logical Right.

LSL - Logical Shift Left. ASR Arithmetic Shift Right.
LSR - Logical Shift Right ASL Arithmetic Shift Left.
Shift ins can be applied to the second Operand.

Eg:- ADD R0, R1, R2 LSL #2 (#2 means 2 Shifts).

Logical Instructions

Negative - No meaning; Zero - Result is all zeroes.
Carry - After Shift it was left in the Carry flag.
Overflow - No meaning.

Eg :- ADD R0, R1, R2 (Shift left → * by 2^n / Shift Right → * by 2^n).
LSR R2
↓
Shift the contents of R2, R2 times to the Right.

ASL ≡ LSL.

ASR replaces 0's if the value is +ve. ASR inserts 1's if the value is negative.

Rotate Right (ROR):

LSB becomes MSB and each shift the LSB becomes MSB.

Rotate Right extended with C :-

Similar to ROR.

Indirect addressing.

The given value is an addr using which value has to be fetched

Move Instructions :-

MOV, MVN (Move with Negation).

One's complement of the operand is moved.

Load and Store

LDR - Load Register. Eg :- LDRH, LDRB.
STR - Store into Register Eg :- STRH, STRB.

Both are half word store instructions.

Register Indirect Addressing mode :-

Eg :- LDR R0, [R1, -R2]
The address is $R_1 - R_2$.

Pseudo-Ops

Eg: ADR r1, FOO, Location 0x100 is given the name FOO.

An address can be given a user-specified name. That name can be used to load the address into operations.

C:- x = (a+b) - c.

Operands must be brought to registers one of the operands cannot be a memory location.

Assembler:-

```
ADR r4, a.
LDR r0, [r4]
ADR r4, b.
LDR r1, [r4]
```

H.W.

y = a*(b+c+d).

ARM7 Conversion.

```
ADR r4, b.
LDR r0, [r4]
ADR r4, c
LDR r1, [r4].
ADR r4, d
LDR r2, [r4]
ADD r3, r0, r1,
LDR r0, [r3]
ADD r3, r0, r2
LDR r1, [r3]
ADR r4, a.
LDR r2, [r4]
MUL r0, r1, r2.
ADR r1, [r0]
STR r0, [r4]
```

(ADD 2 times)

```
x = (a<<2)|(b&15);

ADR r4, a;
LDR r0, [r4];
MOV r0, r0 LSL2;
ADR r4, b
LDR r1, [r4];
AND r1, r1, #15;
ORR r1, r0, r1;
ADR r4, x;
STR r1, [r4];
```

Addressing modes refer to the way in which the operand for an instruction is specified.

Base plus offset addressing:-

Eg: LDR r0 [r1, #16];

r1 is the base, the immediate value is the offset.

When the offset is immediate, it may have value upto 4096. → only 12 bits are used for specifying immediate operand's value maximum as $4096 = 2^{12}$.

(R1 is not updated with the new address).

Auto Indexing

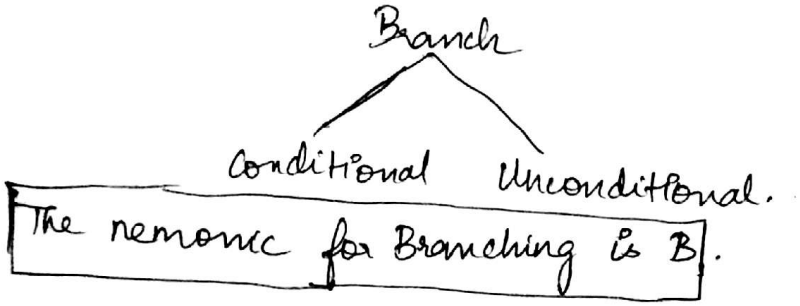
! → indicates auto indexing. Points to a certain value in a range of memory location. Increments the base with immediate operand first and uses the new address value. Eg:- LDR R0, [R1, #16]!;

Post Indexing :-

Eg :- LDR R0, [R1], #16;

fetches the location and then calculates offset.

ARM flow of control



The nemonic for Branching is B.

Syntax :- B <~~address~~ condition> {label}.

B {label}

PC-relative are ↳ Target.

Branches. The branch specifies the offset from the current PC value of the branch target. The offset is in words, but because the ARM is byte addressable, we multiply by 4 (or shift left by 2) to form a byte address.

Eg :- B #100; (400 is added to current PC value)

Conditional Branching :-

Eg :- EQ - equals zero } Labels / Condition.
NE - ≠ 0.

If (a ≥ b) { x = 5; y = c + d; } else x = c - d;

```
:-   ADR  R4, a;
     LDR  R0, [R4];
     ADR  R4, b;
     LDR  R1, [R4];
     CMP  R0, R1;
     BGE  fblock;
```

true block

```
MOV  R0, R0, #5;
ADR  R4, x
STR  R0, [R4];
ADR  R4, c;
LDR  R3, [R4];
ADR  R4, d;
LDR  R5, [R4];
ADD  R3, R3, R5;
ADR  R4, y;
STR  R3, [R4];
```

false block

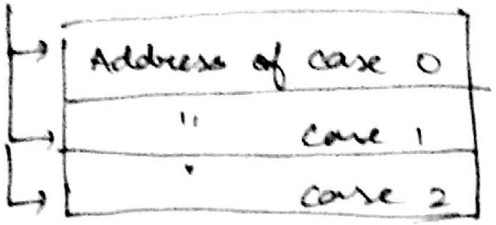
```
ADR  R4, c
LDR  R0, [R4];
ADR  R4, d;
LDR  R1, [R4];
SUB  R0, R0, R1;
ADR  R4, x;
STR  R0, [R4];
```

B after; (Branch around false block)

21/07/2017

Switch (test); → C Statement.

Switch table :-



eg :-

```
ADR r2, test ;
LDR r0, [r2] ;
ADR r1, switchtab ;
ADR r1, [r1, r0, LSL #2]
switchtabl DCD r1 ;
```

FIR filter :-

```
for (i=0 ; f=0; i<N; i++)
    f = f + c[i] * x[i];
```

Looping → Incrementing. / Decrementing.

eg :-

```
MOV   r0, #0 ;
MOV   r8, #0 ;
ADR   r2, N ;
LDR   r1, [r2] ;
MOV   r2, #0 ;
```

Ass. 2

i) (i=0 ; i<20 ; i++)
(x[i] = a[i] * b[i])

ii) for (i=0; i<10; i++)
{for (j=0; j<10; i++)
{z[i] = a[i,j] + b[i]};

iii) ARM code for

if (x-y>3)
{a = b - c; a = 0;}
else
{y = 0; d = e + f + g;}.

iv) C assignments

z = a+b
g = (e-d)+(e-f)
z = a * (b+c) - d*e

Branch & Link

Used in cases of Subroutines.

Eg BL foo → Direct transfer
BL <cond> Sub-routine-label
↳ Based on a Condition.

While Branching the addr of the next instruction must be saved. This is done using r14 → Link Register.

Nested functions & Recursive functions

We use a stack to keep track of the function calls.

Activation Record

Entire stack frame that is pushed. contains Return values parameters and the Register Contents.

Procedure Linkage :-

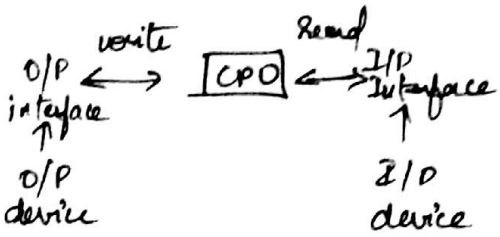
Conventions that are passed in and out of procedures.

14/07/2017

Programming Input & Output

→ The Interface between the CPU and I/O devices is a set of registers.

→ CPU communicates the data via the registers.



Data Registers :- Stores the data.

Status Registers :- Purely Read only and CPU reads the status of execution.

8251 UART :- (Universal asynchronous receiver Transmitter).

→ Provides Serial Communication.

→ These days a larger chip is used.

→ Used for transmission accordingly.

H/W :- Comm. of I/O devices

Serial Communication

→ Data transmission happens with a linear flow.

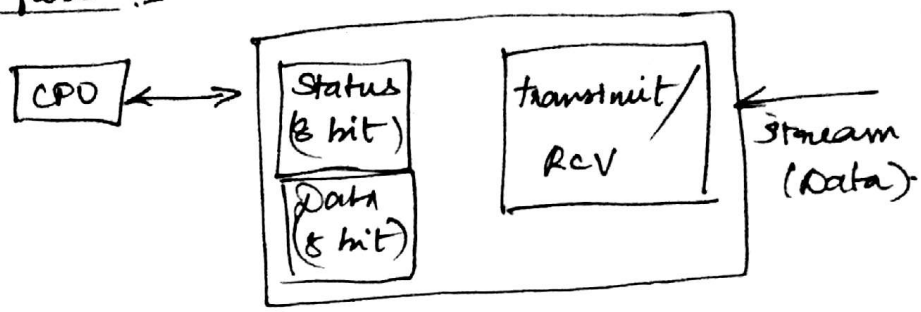
Parameters :- Baud (bit) rate.

→ Number of bits / character.

→ Parity.

→ Length of stop bit (1, 1.5, 2 bits).

Data flow :-



I/O programming :-

- Mapped I/O
- Memory-mapped I/O
- I/O-mapped I/O. → Specially mapped addresses.
- Special instruction I/O.
- Intel x86 architecture (in, out).

x86 provides a separate space for I/O.
OUT - Sending data to Control Register.
IN - Sending data out of Control Register.

Memory Mapped I/O :-

→ Normal Read & Write Instructions.
→ Address space is shared between I/O devices and memory.
→ In architecture, it is given by LOAD and store.

Arm memory - mapped I/O

→ Define a location for the device.
→ The Read/write code.
→ Object code is not created for pseudo Operations.

High level languages

→ Peek() and Poke()
→ Peek is used for reading a memory location.
→ Poke is used for writing into a memory location.

peek returns a value and takes in location. poke takes in location and the value to be stored.

Data transfer

→ I/O devices are slower than CPU.

→ To check the status of the device.

→ Polling (busy-wait)

→ Interrupt.

Polling :- Processor is in a loop and checks if device is free or not. The method is very inefficient.

Interrupt I/O :-

→ Signals raised by devices to the CPU.

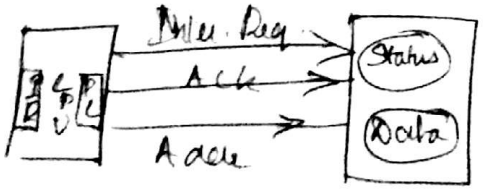
→ the program Counter's value is changed to point on ISR.

→ The last address value is stored.

27/2017

⇒ Busy/wait is very inefficient.

→ Interrupts allow a device to change the flow of control in the CPU.



Interrupt Behaviour

→ Based on subroutine call mechanism.

→ Interrupt forces the subroutine call to be the next Instruction.

Interface :-

→ CPU and device enter a handshake.

→ Acknowledges the time when the Interrupt can be handled.

→ Device asserts interrupt Request.

→ An interrupt has to be fully handled before another one can be handled.

→ PC is set to the beginning of the Interrupt Service Routine.

Priorities :- Determines the order of precedence for Interrupt based on priority.

Vectors :- allows the interrupting device to specify a handler.

Changing Priority

→ Connect a device to a different Location by hardware modification.

Masking :- Interrupt with priority lower than current priority is not recognized until pending interrupt is complete

Non-Maskable Interrupts

highest priority, never masked.

Eg :- During Power failure.

When Several devices have the same priority we use an OR gate to take in all their Interrupts.

Vectored Interrupts

→ Vectors provide flexibility.

a) The ability to define the interrupt handler that should ~~receive a~~ service a request for a device.

Interrupt Sequence

→ Request
→ Acknowledge
→ Request sent
→ Index to table
→ Address of the handler.
→ Called
→ Restored.

Adv

→ Handler depends on Vector Number.
→ Arbitary relations between devices & handler.

Overhead

→ Execution.
→ Mechanism.
→ Pipeline penalties
→ Cache penalties.

ARM7 Interrupts

→ Fast Interrupt
→ Interrupt Requests.

Assignment
FIQ
IRQ

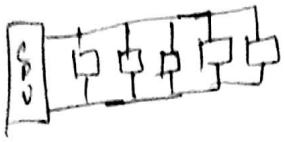
6/02/2017

UNIT - II

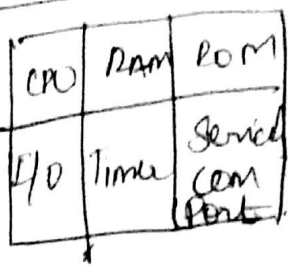
20/07/2017

MICROCONTROLLERS AND EMBEDDED PROCESSORS

Microprocessor



MicroController



Application Development

Microprocessor

→ Costlier & bulkier.
→ Versatile features.

Microcontroller

→ Reduced cost, space power.

Embedded Systems

→ Made for one task only

Personal Computer

→ Can perform several tasks.

Criteria for a MicroController

→ Cost
→ Memory capacity.
→ Peripherals.
→ Power Consumption.
→ Speed
→ How easily they can be upgraded.

Manufacturers

→ Intel
→ Atmel
→ Phillips
→ AMD.

8051 Microcontroller :-

→ Intel manufactured it.
→ 4 I/O ports, 1 serial ports.
→ 128 bytes of RAM.
→ 6 Interrupt Sources.

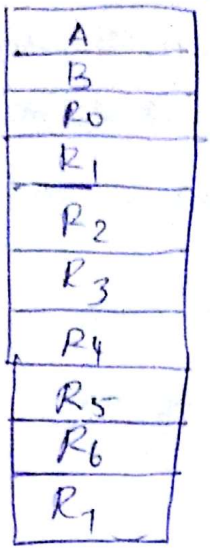
The variants of the 8051 are 8052 and 8031.

Register Set of 8051

Register are used used to store info temporarily.

Register Set

Eg:- MOV
destination, source



#01F7H.
↳ Indicates F is a hex Number.

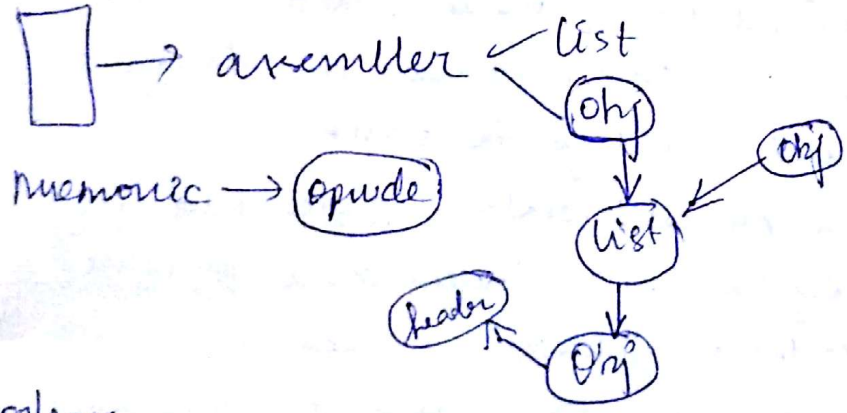
ADDA, source ; ADD is the source operand to the accumulator.

Assembly language is of mnemonics.

ALP → ALSS [Directives/pseudops].

Once END is reached No more pseudops will be executed.

SOS is kept at 0000 with regards to its PC value.



21/01/2017

SJMP → short Jump.

24/01/2017

Assembler directives are not converted into machine code.

The DB (Define data)

→ Reserve Storage space
→ Giving names to memory location.

Label :- Name DB value.

Any value is internally stored as hexadecimal. The input types are decimal, binary, hexadecimal, ASCII. The converted hex values are suffixed H, B for hexa and binary.

ORG <value> → Starts the execution from this point.

A - 65 ASCII ↳ decimal / Hexa.

a = 97 ASCII

END statement is the last statement of the Assembly language code.

EQU - equate Label EQU <value>. Used for assigning a value or a constant. There is no specific space assigned so it differs from DB in this aspect.

Program Status Word / Flag Register :

→ 8 bit register (PSW).

→ 6 bits are only used.

The four conditions involved are Cy (CARRY), AC (auxiliary carry), Parity even (P), and OV (overflow). The two unused bits are userdefinable. PSW3 and PSW4 are designed as RS0 and RS1 and used to change the bank.

Out of the 128 bytes we call certain bytes and as Register banks.

PSW4	PSW3	
0	0	RB_0
0	1	RB_1
1	0	RB_2
1	1	RB_3

Registers can be called either by name or their address. The default Register bank is RB_0. RB_1 serves the purpose of stack area.

Eg:-

00111000 (38) Cy = 0 (No carry after D7)

① 0010 111 1 (2F) AC = 1 (D3 to D4 carry)

0110011 1 (67H) P = 1 (odd no. of ones)

1001 1100 (9C) Cy = 1 (1 after D7).

② 0110 0100 (64) AC = 1 (D3 to D4).

1000 0000 P = 0 (even no. of one's).

(8) (BB+93)

1000	1000	CY = 1
1001	0011	AC = 0
0001	1011	P = 0 (even no. of 1's).

Split-up of RAM

00 - 1FH	Register Banks & Stacks	32 bytes.
20 - 2FH	Bit addressable read/write memory	16 bytes.
30 - 7FH	Read and write storage/Scratch pad	80 bytes.

00 - 07 → Register Bank 0.
08 - 0F → Register Bank 1.
.
.
.

Each Register Bank has 8 Registers R0 - R7.
Register Bank 0 is by default.

26/07/2017

Stack

→ It is a section of RAM.
→ Used by the CPU to store information temporarily.
→ Register used to access the stack is called the SP (Stack Pointer) register.
→ When powered on, the default loaded value is 07.

PUSH :

→ Storing of a CPU register in the stack.
→ Pushing data into the stack, SP is incremented by one.
(Increments SP & places the value).

Pop :-

→ Loading contents in stack into a register.

→ SP is decremented once.
(Takes the value & decrements SP).

PUSH AND POP should use address of Registers

CALL :-

→ The stack is used to store the address of the next instruction.

Eg :- PUSH <value>, POP <value>.

Do not use Bit addressable area.

1) Show the contents of PSW after the execution of foll.

Instructions :-

```
MOV  A, #0BFH
ADD  A, #1BH.
```

2) Show the status of C, AC, P Flags after the '+' of 9CH. and 6AH

3) Use assembler directives to place 0FCH, 05H, 76H, 28D and the character string 'Sam' in consecutive memory locations beginning from location 50H.

Ans :-

```
ORG #50H.
DB #0FCH, #05H, #76H, #28D.   (Convert to Hex in Memory map)
DB "Sam".
END.
```

50H	FC
51H	05
52H	76
53H	1C
54H	53
55H	41
56H	4D.

② 9CH → 1001 1100 N11

6AH → 0110 1000 111 1

01 111 0

1001 1100
0110 0100
0000 0000

C = 1
AC = 1
P = 0.

3) Write 8051 instructions to use register bank 3 and load value 05H into registers R_0 and R_3.

Ans:-

```
SETB PSW.3
SETB PSW.4      } Selects RB3.
MOV  R0,#05H
MOV  R3,#05H.
```

4) Write PUSH instructions to PUSH the contents of the registers on STACK after the execution of the foll. instructions.

Ans:-

```
MOV  SP,#4FH.
SETB PSW.3
MOV  R0,#25H
MOV  R1,#0CH.
MOV  R1,#05H
MOV  A,#0CEH.
```

0F → R_7
0E → R_6
0D → R_5
0C → R_4
0B → R_3
0A → R_2
09 → R_1
08 → R_0

```
PUSH 08H
PUSH 09H
PUSH 0AH
PUSH 0E0H
```

RETHOIT

CALL Instructions

→ Call instruction is used to call subroutine.

→ Subroutines are often used to perform tasks that need to be performed frequently.

→ LCALL (Long call).

→ 3byte Instruction.

→ First byte is the opcode.

→ Second and third bytes are used for address of a target Subroutine.

ACALL (absolute call):-

→ 2 byte instruction.

→ 11 bits are used for address within 2K byte range.

RET - marks the end of the subroutine.

→ The processor is notified regarding the end of execution.

NOP → Does not do any operations but some clock cycles are wasted.

Default stack pointer is at 07.

31/07/2017

1) The CPU requires a certain no. of clk to execute an instruction. In 8051, these clock cycles are called as machine cycles. The length of the machine cycle depends on the frequency of the crystal oscillator. The frequency of the crystal oscillator connected to the 8051 family can vary from 4MHz to 30MHz. The original 8051, one machine cycle has last 12 oscillator periods. Therefore to calculate the machine cycle of 8051 we take $\frac{1}{12}$ th of the crystal frequency and then take its inverse. The following has a different crystal frequency.

a) 11.0592MHz.

b) 16MHz.

c) 20MHz.

Ans) a) Crystal frequency = 11.0592 MHz.

$= \frac{11.0592 \times 10^6}{12} = 921.6\,KHz.$

$= \frac{1}{921.6KHz} = \frac{10^{-3}}{921.6} = 1.085\,\mu s.$

b) Crystal Frequency = $\frac{16 \times 10^6}{12} = \frac{4}{3} \times 10^6 = \frac{3}{4} \times 10^{-6}$

$= 0.75\,\mu s.$

c) Crystal Frequency = $\frac{\cancel{16} \times 10^6}{12} = \frac{5}{3} \times 10^6 = \frac{3}{5} \times 10^6$

$= 0.6\,\mu s.$

b) Find out how long it takes to execute each of the following statements for an 8051 of 11.0592 MHz.

	# of m/c Cycles
MOV R_3, #55	1
DEC R_3	1
DJNZ R_2, target	2
AJMP	2
SJMP	2
NOP	1
MOL AB	4

a) 11.0592 = 1.085 μs.

Multiply the delay with no. of cycles and sum it up.

Eg:-

MOV R_3, #250	1	1		1
NOP	1	250		250
NOP	1	250		250
NOP	1	250		250
NOP	1	250		250
DJNZ R_3, H	2	250		500
RET	1	1		2
				1503

=) 1503 x Machine Delay = 1503 X 1.085 x 10^{-6}

= 1630.755 μs.

(2)

	MOV, R_2, #200	1		1
A:	MOV R_3, #250.	1		200.
H:	NOP	2	200 X 250	50,000
	NOP	2		50,000
	DJNZ R_3, H			1,00,000
	DJNZ R_2, A.	200		400
	RET	2	1	2
				217000 μs.

=) 217651 μs.

c) Find the delay with the following program. when the machine cycle period is 0.546 μs.

	MOV R_2, #100	1	1	1
A:	MOV R_3, #255	1	255	255
H:	DJNZ R_3, H	2	255 X 100	51000
	DJNZ R_2, A.	2	100	200
	RET	2.	1	2.
				25096.
				= 0.546

d) Write a prog to toggle all pins of port 2 continuous between the toggling with a delay as 1s. Assume crystal frequency (XTAL) is 22 MHz.
(Find out the no. of noops).

2/08/2017.

max. amount of delay that can be got using a register Pair (or) a single Register.

[Frequency = 11.0592 MHz]

Single Register

```
1     DELAY MOV  R0 #FFH     1
255   DJNZ  R0 , HERE        2
1     RET                    2
```

$\Rightarrow$ ~~# 1110+2.~~ $= 513 \times 1.085 \times 10^{-6}$

$=$ ~~$1113 \times 1.085 \times 10^{-6}$~~.

$= 556605 \times 10^{-6}$

$= 0.55ms$.

Register Pair $= 1.085 \mu s$.

```
1        MOV R0 , #FFH
1 255    MOV R1 , #FFH : AGAIN
2(255)^2 HERE: DJNZ R1 , HERE
2 (255)  DJNZ R0 , AGAIN
1 (1)    RET.
```

$= 1 + 255 + 2(255)^2 + (255)2 + 1$

$= 0.14s$.

The delay can be increased by introducing an additional Register and extra nesting.

Addressing Modes

→ CPU can access data in various ways.

→ Immediate.

a) Source Operand is a constant.

b) Immediate data must be preceded with '#'

→ Register.

a) Registers hold the data to be manipulated.

→ Direct.

a) Register bank locations are accessed by register names.

→ Register Indirect

→ Indexed.

Addressing modes specify where the data is present. Register, direct and Register Indirect are used for accessing the components in memory.

Immediate Addressing Mode :

→ The operand must be preferred with the pound sign, "#".

→ The info can be loaded into 16-bit DPTR registers

→ The 16 bit register can be accessed as two 8-bit registers one being high byte DPH and the other being low byte DPL.

Value = 65535 (FFFFH) → Illegal and can't be stored.

DW → assign a constant value to a label. and the immediately addressed.

Values can be sent to 8051 Ports. DPTR stores the address locations and not the actual value.

Register Addressing Mode:-

→ Use registers to hold data to be manipulated.

→ The source and destination registers must match in size

→ The movement of data between R_n registers is not allowed.

Eg:- Invalid Statements.

MOV R7, DPL (Different size).

MOV R_3, R_4 (Intra transfer between registers)

Direct Addressing mode:

→ The RAM locations from 20 to 7FH are used.

→ There is no # sign with the operand.

00 → 07 → Register banks.
08 → 0F → Default stack.

20 - 2F → bit addressable

The register banks are accessed by their register names. Every Register has a name and an address.

SFR Registers

Special Function Register can be accessed by their names or by their addresses.

→ SFR registers have addresses between 80H and FFH.

→ Not all the addresses in this range are used.

SFR Register symbols are ACC*, B*, PSW*, SP, DPTR, DPL, DPH, P0*, P1*, P2*, P3*, IP*, IE*.

ACC*	accumulator.	0E0H.
B*	B register	0F0H.
PSW*	Program Status Word.	0D0H.
SP	Stack Pointer	81H.
DPTR	Data pointer 2 bytes	
DPL	low Byte	82H.
DPH	High Byte	83H.
P0*	Port 0	80H
P1*	Port 1	90H
P2*	Port 2	0A0H.
P3*	Port 3	0B0H.
IP*	Interrupt priority	0B8H.
IE*	Interrupt Enable	0A8H.

* - Bit addressable.

~~TMOD~~

Eg:- Send 55H to ports P1 and P2 using i) Names ii) Addresses.

① MOV P1, #55H.
MOV P2, #55H.

② MOV 90H, #55H.
MOV 0A0H, #55H.

Stack and Direct Addressing Mode.

Move the contents of registers A, R0, R1 respectively of bank 0 to the registers B, R0, R1 of bank 1 using stack operations.

→ The default stack is Bank 1

B0 { R0, R1 } ⟶ { R0, R1 } B1

MOV SP, #3FH.

~~[illegible]~~.

PUSH 0E0H
PUSH 0
PUSH 1
SETB PSW3

POP D6
POP 09.
POP 0F0H.

What is the address of Register R3 in Bank 2.

Ans :- 12

Register Indirect Addressing Mode.

→ A register is used as a pointer to the data.

→ When R0 and R1 hold the addresses of RAM location they must be proceeded by the '@' sign

Eg: MOV A @R0. Move contents of RAM whose address is held by R0 into

Eg:- Write a program to copy the value 55 into the RAM memory locations 40H to 45H using.

i) Direct
ii) Register Indirect without (loop)
iii) Register Indirect addressing (loop).

i)
```
MOV 40, #55H
MOV 40H, A
MOV 41H, A
MOV 42H, A
MOV 43H, A
MOV 44H, A
MOV 45H, A
```

ii)
```
MOV A, #55H
MOV R0, 40H
MOV @R0, A
INC R0      (R0 = 41H)
MOV @R0, A
INC R0      (R0 = 42H)
MOV @R0, A
INC R0
MOV @R0, A
INC R0
MOV @R0, A
INC R0
```

iii)
```
MOV A, #55H
MOV R0, 40H
MOV R1, #5
LOOP:- MOV@R0, A
       INC R0
       DJNZ R1, LOOP
```

In R.I.A looping is possible. Also data can be accessed dynamically.

a) Write a program to clear 16 RAM locations starting at RAM address 60H.

Ans :-
```
      CLR A
      MOV R1, #60H
      MOV R7, #16
AGAIN: MOV @R1, A
      INC R1
      DJNZ R7, AGAIN
```

5) Take 10 bytes from Data RAM locations 45H - 54H. add 02 to each of them and save the result in Data RAM locations 79H to 70H.

8/8/2017

① MOV SP, #45H. → Value '45H' sent to SP.
MOV SP, 45H. → Value held at 45H sent to SP.

R0 and R1 are used for storing address.

ROM is generally of 4KB.

DPTR → A } Indexed Addressing / MOVC A, @A+DPTR.
"code" → On Chip Rom.

Eg:- The word Sam is burnt in flash ROM starting at location 0400H. Write a prog to read this data into internal RAM locations starting at 60H.

```
Ans:-   MOV  DPTR, #0400H.
①       MOV  R0, #60H.
        MOV  R1, #03H.
HERE:   CLR  A.
        MOVC A, @A+DPTR.
        MOV  @R0, A.
        INC  DPTR.
        INC  R0.
        DJNZ R1, HERE.

②       ORG. 0400H
        DB   "SAM".

        END.
```

⑤ DPTR contains America. Write a program to (H.W.) transfer bytes starting at 40H. Assume that the string uses null character to mark the end. (value = 0)

① MOV DPTR, #DATA.
MOV A, #0FFH. } Port is configured as an I/P. FF means all 1's. For configuring a port as O/P fix all 0's.
MOV P1, A.
BACK: MOV A, P1
MOVC A, @A+DPTR.
MOV P2, A.
STMP BACK.

ORG 300H.
DATA DB 1, 4, 9, 16, 25, 36, 49, 64, 81, 100.

* 4 tables. Multiply lookup table.

③ Write a delay subroutine using RAM location 45H as counter. Use max value of FF. (Instead of DJNZ Instead of RAM location).

Ans:- DELAY: MOV 45H, #055H.
HERE: DJNZ 45, HERE
RET.

Bit Address

20-2F RAM Bit addressable.

PSW, TCON, SCON, ACC.
Port → P0 - P3.
SETB, CLR, CPL, JB } Instructions.

JB bit, target → Jump.
JNB bit, target → Not
JBC → Clear.
MOV → one operand should be carry

Every location from 20-2F is given a specific address.

The last address in 2f ranges from 78 – 7F.

00 – 7F RAM → Direct addressing mode + Register Indirect

Bit addressing address ← 00 – 7F → Used only with Bit manipulation + direct addressing mode.

Pi.X → Can be used to set each byte of the Port Pi

Eg :- $P_1.6$, $P_2.3$.

i → 0 → 3 ; X → 0 → 7.

Bit Memory Map :-

Bit Addresses	Referring to.
00 – 7F	RAM locations (Bit addressable)
80 – 87	Port 0.
88 – 8F.	TCON (Timer Control Reg
90 – 97	Port 1.
98 – 9F	SCON (Serial).
A0 – A7.	Port 2.
A8 – AF	Interrupt Enable register
B0 – B7	Port 3.
B8 – BF	Interrupt Priority Control
C0 – CF	Not Assigned.
D0 – D7	PSW.
D8 – DF	Not Assigned.
E0 – E7.	Accumulator.
E8 – EF	Not Assigned.
F0 – F7	B register.

07/06/2017.

Arithmetic & Logical Instructions

ADD A, source → Register / immediate / memory (direct, indirect).

A = A + Source

All flags → AC, CY, P, OV

ADD A, 45H → (direct)

ADD A, #45H → (immediate)

```
MOV R0, #45H
ADD A, @R0
```
Indirect

Add with carry ADC (ADD WITH CARRY)

ADC A, source

A = A + source.

Considering Carry,

A = A + Source + CY.

Multibyte Addition

BCD → Binary Coded Decimal.

- Packed BCD → 1 byte → 2 BCD nos.
- Unpacked BCD.

↳ 1 byte is used to represent a single BCD no.

Eg 0000 → 0 ... 0000 1001 → 9.

Eg 15 → 0100 0101 Packed BCD.

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

To adjust the result of BCD operation we use DA A.

Eg:- DA A adds 6 (0110) to the lower 4 bits of accumulation if lower 4 bits is > 9 or if auxillary carry flag is set to 1. Higher order bits are also added with 6 if the value > 9. CY = 1.

Arithmetic & Logical Instructions

SUBB A, source → reg / imm / memory.

Mnemonic.

A = A - Source - CY

```
CLR C
MOV A, #42H.
MOV R1, #12H.
SUBB A, R1
```

Subtraction is performed using 2's complement algorithm.
arithmetic.
42 → Minuend.
12 → Subtrahend.
cy = 0; cy = 1.
↓ ↓
Result +ve Result (-ve).
↓
2's Complement form.

Multiplication :-

MUL AB. | Mnemonic

byte × byte → A
Lower byte
Buffer Byte.

Division :-

DIV AB. / 2 Registers. N → A ; D → B.
⇓
R → A ; R → B.

OV = 1 if B = 0.
(Divide by zero).

AND

ANL A, #data ⇒ AND A with data place result in A.

ANL destination, source

ANL A, R_N.
ANL A, direct.
ANL A, @R_0/R_1.
ANL direct, #data.

ANL direct, A

OR

ORL destination, source
destination = destination or Source.

XOR

XRL destination, source
dest = dest XRR source.

CPL A → Complement contents of A.

CJNE → destination, source, relative address.

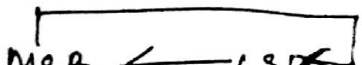
Jump to the target address if source and destination are not equal. It results in the setting of carry flag. ⇒ Destination ≠ Source

else Dest = Source.

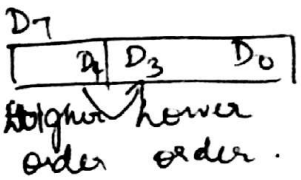
RR A. (Rotate Right accumulator).

Every bit is shifted once from MSB to LSB.

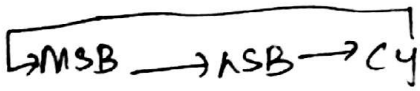
RL A (Rotate left accumulator).



SWAP A (operation accumulator).



RRC A (Rotate Right Carry).



RLC (Rotate left carry)

OV flag set :-

i) When there is a carry out of D3 and nothing out of D7.

ii) Carry from D7 but nothing from D6 to D7.

1) Two numbers are stored in R0 & R1. Verify if S is 7 then

FR. Assume that RAM locations 40 to 44 have the following values.

i) 7D
ii) ED
iii) C5
iv) 5B
v) 30.

Write a program to find the sum of the values at the end of the program register A should contain a low byte and R7 high byte. All values are in hex.