INDIAN INSTITUTE OF TECHNOLOGY BOMBAY
ELECTRICAL ENGINEERING DEPARTMENT

REPLACEMENT TEST-1

Saturday	**EE 309: Microprocessors**	Time: 1430-1645
Apr. 17, 2021	Spring Semester 2020-21	Marks: 40

Marks will be scaled depending on the tests being replaced.
Code should be clear and well commented.

Q–1 **a)** For each of the following 8051 instructions, specify if the given usage is correct or not. If it is correct, specify what the instruction will do. Otherwise state why the usage is illegal.

```
a)  MOV    R1, R7
b)  SUB    A, @R0
c)  MUL    AB
d)  CJNE   B, #21, label
e)  XCHD   A, B
f)  JMP    @A+DPTR
h)  MOV    A, @R3
i)  DJNZ   @R0, label
```

Soln. 1-a)

a)MOV R1, R7
Incorrect: both arguments cannot use R registers
b)SUB A, @R0
Incorrect: There is no subtract instruction without borrow.
Only SUBB is available for 8051.
c)MUL AB
Correct. Contents of A and B are multiplied and the result is placed in A (lower byte) and B (upper byte). Overflow flag is set if the result does not fit in a single byte.
d)CJNE B, #21, label
Incorrect: a direct address cannot be compared to an immediate.
The allowed comparisons are accumulator/Rn/@Ri with an immediate value and accumulator with a direct address.
e)XCHD A, B
Incorrect: XCHD allows only one addressing mode: exchanging the lower nibble of the accumulator with that of an indirectly addressed byte (XCHD A,@Ri).
f)JMP @A+DPTR
Legal. The contents of DPTR and A are added and the sum is loaded into PC. For example, if DPTR = 03C6H and A contains 58H, the program will jump to location 041EH.
g)MOV A, @R3
Incorrect: Only R0 and R1 can be used as pointers.
h)DJNZ @R0, label
Incorrect: Only R registers or direct addresses are allowed as operands to be decremented.

– [4]

b) Write a code fragment in 8051 assembly language using bit addressing instructions to evaluate

$$\mathrm{P3.4} + \mathrm{F0} \cdot (C \oplus \overline{B})$$

Where C is the value of carry flag and B is the value stored at the most significant bit of byte address 28H. If it evaluates to '1', P3.5 should be complemented.

Soln. 1-b) 8051 provides logical AND and OR functions for single bits, but not XOR. Therefore the given expression should first be simplified to use AND and OR.

$$C \oplus \overline{\mathrm{B}} = C \cdot B + \overline{C} \cdot \overline{B}$$

We use F1 in PSW (PSW.1) as a temporary bit variable. We need one more temporary variable for this evaluation, for which we can use any available bit addressable location – say 20H.0 or A.0.

```
org 0000H
ljmp Init

ORG 0100H
Init:
; Data set up
SETB C
SETB F0
SETB 24H.7
; Now compute the expression
MOV PSW.1, C ; Save C in user flag
ORL C, 24H.7    ; C = B OR C
CPL C           ; C = Bbar.Cbar
MOV 20H.0, C    ; Save this value
MOV C, PSW.1    ; Recover original C
ANL C, 24H.7    ; C = C.B
ORL C, 20H.0    ; C = C.B + Cbar.Bbar = C XOR Bbar
ANL C, F0       ; C = F0.(C XOR Bbar)
ORL C, P3.4     ; C = desired value
JNC Done ; Do nothing if False
CPL P3.5 ; Complement if True
Done: sjmp Done
END
```

– [4]

c) A program calls a function written in 8051 assembly language with a number between 0 and 3 (both inclusive) stored in register A. We want the called function to execute the corresponding code fragment (starting with labels L0, L1, L2, L3) and then return to the main program.
Write the assembly language code for the called function, by using a jump table consisting of an array of AJMP instructions.
(For purposes of illustration, let the code fragments labelled L0, L1, L2 and L3 replace the value in A by the byte read from ports P0, P1, P2 or P3 respectively).

Soln. 1-c)

```
org 0000H
ljmp Init       ; To test program

ORG 0100H
Init:           ; Data and test set up
MOV A, #02 ; Test with A = 2. Change later.
ACALL SwitchFn ; Call the function
Done: sjmp Done ; Must return here.

SwitchFn:
ADD A, A        ; Offset = 2*A
MOV DPTR, #JmpTab
JMP @A+DPTR     ; to selected AJMP
JmpTab:         ; Jump table
AJMP L0
AJMP L1
AJMP L2
AJMP L3
RET             ; Will never come here
L0: MOV A, P0   ; Port to A as examples
RET
L1: MOV A, P1
RET
L2: MOV A, P2
RET
L3: MOV A, P3
RET
END
```

– [**4**]

Q–2 a) What is the advantage of PC relative addressing? Give examples of instructions of 8051 which use PC relative addressing.

Soln. 2-a) PC relative addressing makes the code position independent. It will work the same irrespective of where it is loaded in the code memory space. This is because addresses are not absolute – if the program is executing at a different address, the value in PC will reflect that and the offset from PC will always be the same. Since PC always points to instructions, PC relative addressing applies only to code memory, not to RAM.

In 8051, the short jump instruction sjmp and all conditional jumps are PC relative. Also, the data movement instruction:
MOVC A, @A+PC
is PC relative. – [**2**]

b) How does decimal adjust work after addition in 8051? Show how it produces the correct decimal result after a binary addition.

Soln. 2-b) The difference between binary and decimal addition is that in one case, we use the base sixteen for each nibble, while the other uses the base of ten. The maximum sum of two decimal digits is 18. If the binary sum does not exceed 9, no correction will be required. If the binary sum exceeds 9, we need to correct it. The need for correction is indicated by either the sum digit being between A and F (> 9) or if it is 0, 1 or 2, by the fact that a carry resulted from this nibble. In this case, the solution is to add 6 to the digit, so that the hex digit A goes to 0 with a carry, B goes to 1 with a carry etc. If carry is already set, a 0 in the binary sum will go to 6 (with carry it represents 16), 1 to 7 (representing 17) and 2 to 8 (representing 18).

So the decimal adjust works by adding 6 to the lower or upper nibble if either its value is > 9 or if an Auxiliary carry resulted in case of the lower nibble and carry resulted in case of the upper nibble.

Take the example of adding 79 and 38. A binary addition will produce B1 as the result with Auxiliary carry set. On decimal adjust, 6 will be added to the lower digit, producing 7. Since the upper nibble is > 9, 6 will be added to it also, producing 1 with carry. Thus the number becomes 17 with carry set – representing 117, which is the correct result. – [**2**]

c) Under what conditions is the overflow flag in PSW of 8051 set after add/subtract and after mul/div instructions.

Soln. 2-c) After addition/subtraction, the overflow flag is set if the result will produce the wrong sign due to its magnitude exceeding the representable range for *signed* numbers.
For example, 73H (115 decimal) and 35H (53 decimal) are legitimate signed numbers. Adding these will produce A8 as the result. Interpreted as an unsigned number, this 168 and the correct value. However, interpreted as a signed number, the most significant bit is set and hence, it represents -88. Thus adding the legitimate signed numbers 73H and 35H has produced the wrong result, because the magnitude of the sum is > 127, which is the highest representable positive number as a byte.
8051 detects this case as the XOR of carry into and out of the most significant bit and sets the Overflow flag if is true.
In case of multiplication, The result is 2 bytes with the more significant byte of the product stored in B and the less significant byte in A. Overflow cannot occur as the result cannot be more than 16 bits. However, OF flag is set if the result is > 255 and cannot be contained in a single byte, necessitating storage of the upper byte in B.
After division, OF is set if a division by 0 is attempted. – [**2**]

d) What does the parity flag in 8051 indicate? When is it updated?

Soln. 2-d) Parity flag in 8051 is '1' if the contents of the accumulator have an odd number of 1's, '0' otherwise. It always reflects the status of A, so it will be updated whenever the value of A changes. – [**1**]

e) The 8052 has 256 Bytes of internal RAM, of which the top 128 occupy the same address space as the special function registers. Show how these are distinguished by using an example in which you read a byte from the memory address 80H and write it to a special function register at the address 80H.

Soln. 2-e) SFR's are referenced whenever direct addressing is used. RAM is accessed if indirect addressing is used. Since push and pop use the stack pointer, these will use the RAM if SP value lies between 128 and 255. Similarly, indirect reference through R0 or R1 will reference the RAM. Direct addresses will reference the special function registers.
In the given example,
MOV R0, #80H ; Load address for indirect reference
MOV A, @R0; Read memory
MOV 80H, A; Write to SFR (Port) – [**1**]

f) A system with 8051 processor has an external ROM connected to it. It encounters the instruction MOVC A, @A+DPTR
How does it decide whether the constant is to be fetched from internal ROM or from external ROM?

Soln. 2-f) This depends on the value of the address as well as the status of the $\overline{\text{EA}}$ Pin.
For addresses between 0000H and 0FFFH (both inclusive) internal ROM is accessed if $\overline{\text{EA}}$ is high, external ROM if this pin is low.
For addresses above this range (1000H upwards) external ROM is always accessed since the internal ROM is only 4KB in size, with a maximum address range of 12 bits. – [**1**]

Q–3 a) What problem can occur if one wants to read the count from a running 16 bit timer in 8051? Show how this problem can be avoided.

Soln. 3-a) Since the current count is in two bytes, we have to read these sequentially. This presents a problem – because by the time we go to read the second byte, the first one may have changed.

For example, suppose the count is 04FF. We do:
```
        MOV R0, TL0
```
and get FF in R0. However, one machine cycle has passed in executing the MOV instruction. So the count reaches 0500. Therefore, if we now read the high byte by:
MOV R1, TH0; we shall get 05 in R1. Thus we can be misled into thinking that the count is actually 05FF, which is a large error.
(This problem had been tackled in other popular counter/timers like 8254 through the "Latch Counter" command – however, this is not available in 8051).

We can solve this problem by reading the high byte of the timer, then reading the low byte, and finally reading the high byte again. If the high byte read the second time is not the same as the high byte read the first time, one must repeat the cycle. In code, this would appear as:
```
READT0:
        MOV A, TH0
        MOV R0, TL0
        CJNE A, TH0, READT0
```

This may introduce an error of a few cycles – but not the large error described earlier.

Another way is to freeze the counter by clearing its RUN bit, reading it and then re-enabling the run bit. – [**3**]

b) How can we use the timer T0 as a counter for external events? (Give all the hardware and software requirements). What is the highest rate of events which can be counted?

Soln. 3-b) T0 can be used as an event counter (where it counts the number of negative transitions on the pin meant for P3.4.

Additional functions of Port 3 lines

Port Line	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0
Function	$\overline{\text{RD}}$	$\overline{\text{WR}}$	T1 in	T0 in	$\overline{\text{INT1}}$	$\overline{\text{INT0}}$	TxD	RxD

In hardware, we have to use a circuit which ensures that each event causes a negative edge and connect it to Pin P3.4 of 8051. In software, we should configure T0 to act as a counter.

TMOD register at BYTE address 89H								
Bit No.	7	6	5	4	3	2	1	0
Timer:	T1				T0			
Bit Name	G1	C/T1	T1M1	T1M0	G0	C/T0	T0M1	T0M0

(TMOD is not bit addressable). Function of T0 as a counter or timer is selected by setting or clearing the bit 2 of TMOD at the address 89H. To program it as a counter, bit2 of TMOD should be set. Since this register is not bit addressable, bit2 can be set without disturbing other settings by ORing it with 04F.

So in software, during initialization of the system, we have to include
ORL 89H, #04H ; set bit 2 of TMOD
as a statement. The initial count can be cleared by writing 0s to TH0 and TL0. Now P3.4 will be sampled once every machine cycle (12 clock cycles). The count is advanced when a negative step is noticed on the line:
this involves sampling a high level in one cycle and a low one on the next. Since each machine cycle takes 12 clock cycles, the fastest event counting rate is clock frequency/24.
– [**3**]

c) We want to implement a real time clock using an 8051 based system with a crystal frequency of 11.059 MHz. The clock display should be updated every second. This delay is obviously too long to implement using a timer directly. Show how to set up an interrupt service routine for timer 0 such that it calls a function ClockUpd every second as accurately as possible.

Soln. 3-c) The maximum delay possible using a timer in this system is:

$$\text{Delay}_{max} = 2^{16} \times 12 \times \text{Clock Period} = \frac{2^{16} \times 12}{11.059 \times 10^6} = 71.1124 \text{ ms}$$

Therefore we have to use a software counter, incrementing it every time the timer times out and call ClockUpd when the counter reaches a particular value. The count in the timer as well as the software count should be integers. If the timer count is a 16 bit integer m, while the software count is a byte wide count n, we must have

$$1\text{second} = \frac{m \times n \times 12}{11.059 \times 10^6} \qquad \text{So } m \times n = 11.059 \times 10^6/12 = 921583.3$$

Since some time will be taken in incrementing the software counter and calling the function, this count should be adjusted downward to account for that overhead. (Notice that 921583 is also the number of instruction cycles of 12 clock periods each in a second).
Minimum value of n is $921583.3/2^{16} = 14$.
(Lack of compensation for software overhead in your answer will not lead to deduction in marks).

Let us estimate this delay. Every time the timer overflows, it will cause an interrupt. The service routine will do the following:

```
       PUSH   ACC             ;Save registers
       PUSH   PSW             ;
       CLR    TR0             ;Stop timer 0
       MOV    TH0, #HiCount   ;Re-load T0
       MOV    TL0, #LowCount  ;
       SETB   TR0             ;Restart T0
       MOV    A, SwCount      ;Fetch current SW counter value
       INC    A               ;Increment it
       CJNE   A, #n, done     ;If not yet max value
       LCALL  ClockUpd        ;If max value, call ClockUpd
       Clr    A               ;Reset SW counter
done:  MOV    SwCount, A      ;Save it
       POP    PSW             ;Restore registers
       POP    A
       RETI                   ;And return
```

This can be estimated to be about 26 instruction cycles. This overhead will take place at every interrupt – that is n times per second. We can choose n to be 14. Then the overhead can be estimated to be $\approx 26 \times 14 = 364$ instructions.

Choosing m and n to be 2^16 and 14 respectively will give $m \times n = 917540$, which is about 4080 cycles short of the required delay. A better choice is to have m = 46053 and n = 20, which gives $m \times n = 921060$. The software overhead will now be 520 instruction cycles. With the overhead of 520 added to the timer delay, the total delay will be 921580 cycles, which is quite close to the required value. So we choose m = 46053 and n = 20.

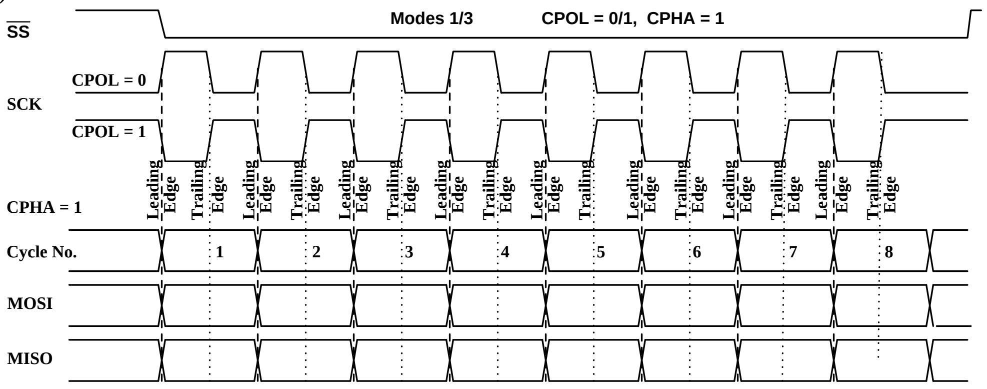
Of course the count to be loaded in T0 should be $2^16 - 46053 = 19483$, such that T0 will overflow after 46053 cycles. Therefore the loaded count should be 4C1BH.

Thus HiCount = 4CH and LoCount = 1BH.

With these choices, timer delay per interrupt = 46053 cycles, software delay per interrupt = 26 cycles, so total delay per interrupt = 46079 cycles. With 20 software iterations, this will give a total delay of $46079 \times 20 = 921580$ cycles. Each cycle has a duration of $12/(11.059 \times 10^6 = 1.08509\mu$s. Thus total delay is 1 second with an error of less than 4 parts per million. – [**4**]

Q–4 a) Show the timing diagram for placing data and latching it in an SPI interface with CPOL = 0 and CPHA = 1 (mode 1). Is it possible to connect a device using mode 1 with another using mode 3? If so, how?

Soln. 4-a)



Mode 1 can be converted to mode 3 by inverting the clock. – [**3**]

b) What is the addressing mechanism used in I2C interface? Give the sequence of signaling by the master in order to address a slave device in order to write to it and to read from it.

Soln. 4-b) In I2C protocol, 7 bit addresses are commonly used, though 10 bit addresses are also supported.

Suppose the micro-controller is the master. It generates the 'Start' message on the bus by pulling the SDA line low while SCL is still high and then sends 8 bits on the serial data bus (most significant bit first), placing new bits on the data line at the downward transition of the clock. These 8 bits include the 7 bit address of the slave device and a R/W bit as the least significant bit. This bit is 0 for a write operation by the master and 1 for read operation. After the master has sent these 8 bits on the data line, its driver transistor goes off during the ninth bit time. The slave device pulls the data line 'Low' during this time to acknowledge successful receipt of the signal. (Recall that this is possible because of the open drain connection). This is called the 'Ack' message.

I2C bus can also use a 10 bit address. This works similar to the 7 bit address, but the first byte sent by the master contains '11110XXY' where the combination 11110 signals the use of a 10 bit address. XX are the two most significant bits of the 10 bit address, while Y is the read write bit as in the case of 7 bit addresses.

The next byte sent by the master contains the remaining 8 bits of the 10 bit address. – [**2**]

c) When is clock stretching used in an I2C interface? How is it implemented?

Soln. 4-c) Clock stretching is used by a slave device to slow down the data transmission rate. An addressed slave device may hold the clock line (SCL) low after receiving (or sending) a byte. (This is made possible by the open drain driver design). This indicates that it is not yet ready to process more data.

The master that is communicating with the slave must wait until the clock line is released by the slave. It waits until it observes the clock line going high, and for an additional minimal time (4μs for standard 100 kbit/s speed) before pulling the clock low again.

Clock stretching is the only time in I^2C bus specification where the slave drives the SCL line.

In principle, clock stretching can be used during the transmission of any bit. In practice, it is used most commonly in the interval just before or after the acknowledge bit. After receiving a byte, the listener needs to check whether the byte was received properly and whether to send an 'Ack' or a 'Nack'. This checking may require more than a bit time.

In this case, the listener can stretch the last bit, while it decides whether to send an 'Ack' or a 'Nack' and then release the clock. The SMB specification limits the time for which a clock period may be stretched. – [**2**]

d) Between SPI and I2C, which serial interface can run faster and why? In the case where multiple slave devices should be connected to a master, which of the two interfaces requires lower pin count and less pc board area? (Give reasons)

Soln. 4-d) I2C bus uses open drain drivers. Pull up is through a common off-chip passive device. This makes it slow. SPI lines are actively driven and can reach much higher speeds.
On the other hand, SPI needs a separate slave enable line for each slave device, which increases the pin count and tracks on the PCB. I2C does addressing through the data line itself – so even for a multiple slave device case, it can manage with just two wires. – [**2**]

Paper Ends