CmpE213 – HW4 8051 Instruction Set
Solution

Due: Tuesday Oct. 14, 2003
Show your work for full credit.

1. Write an instruction (or set of instructions) to:

   (a) Exchange the value of the accumulator with the value in R3.

   ```
   XCH A, R3
   ```

   (b) Set the current register bank to bank 1 (be careful - is the PSW bit-addressable?).

   ```
   CLR RS1
   SETB RS0
   ```

   (c) Jump backward 31H bytes if the carry flag is set.

   `JC CD` (`JC CF` is also acceptable, as the question was vague)

   (d) Clear the accumulator using only a 1-byte instruction.

   ```
   CLR A
   ```

   (e) Increment the value of the DPTR.

   ```
   INC DPTR
   ```

   (f) Jump forward 19H bytes if the accumulator is not equal to 1BH.

   `CJNE A, #1BH, 19H` (`CJNE A, #1BH, 21H` is also acceptable, as the question was vague)

   (g) Perform a bit-wise logical XOR between the bits in the accumulator and the bits at interna1 memory location 42H.

   ```
   XRL A,42H
   ```

   (h) Push the value #65H onto the stack.

   ```
   MOV A, #65H
   ```

```
PUSH A
```

(i) Decrement the value of R1 and jump forward 17 bytes if the result is not zero.

`DJNZ R1, 11H` (`DJNZ R1, 09H` is also acceptable, as the question was vague.)

2. Different, addressing modes can be used to accomplish the same result. For each addressing mode (direct, indirect, immediate, register), write an instruction that loads the accumulator with the value #65H. For each instruction, list precisely which addressing modes are used. For instance, MOV 2AH, #42 uses direct and immediate.

Assume we execute the following instructions before any of the numbered instructions below:

```
MOV 55H, #65H
MOV R0, #65H
MOV R1, #55H
```

1. Register (implied, A) and immediate: (#65H)	`MOV A, #65H`
2. Register (A), direct (55H):	`MOV A, 55H`
3. Register (A), indirect (@R1)	`MOV A, @R1`
4. Register (A and R0)	`MOV A, R0`

3. Assume you are to write a jump instruction that will be at address 4170H in code space. Show a jump instruction for each addressing mode (relative, absolute, long) that jumps to code memory location 400BH. Give the opcode for each instruction.

Relative:

4170H + 2 + reladdr = 400BH => reladdr = -167H < -128H

You cannot write a single relative jump instruction to jump –167 bytes from the current PC. Relative jump instructions use only one byte to encode the relative address, so it can range from –128 to 128 only. You could use a sequence of two relative jump instructions to accomplish the task.

Absolute:

An absolute jump can only be performed if the source and destination of the jump are within the same 2K page of memory, which they are in this case.

The source instruction is at 4170H = 0100 0001 0111 0000 B

After incrementing the PC twice (for a two-byte instruction), we will have the source PC:

4172 H = 0100 0001 0111 0010 B

The destination PC is 400B H = 0100 0000 0000 1011

The source and destination PC addresses share the five most significant bits, hence we can use an absolute jump to go from one to another.

The two-byte opcode is formed by taking the 11 least significant bits of the destination address (A10-A0) and placing them in the following positions:

A10 A9 A8 0 0 0 0 1 A7 A6 A5 A4 A3 A2 A1 A0

0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 1 = 010BH

The corresponding mnemonic will be: `AJMP DEST`, where DEST is the label for the destination instruction located at 400BH in code memory.

4. For the following code.

```
start:      MOV 2AH, #5
            MOV R0, #80H
            CLR A
loop:       MOV @R0, A
            INC R0
            DJNZ 2AH,loop
stop:       SJMP stop
```

a) Find the final value of any registers or memory changed by executing this code segment.

The final values are:

A = 00H, as the only instruction affecting A is `CLR A`.

R0 = 85H as it is initialized to 80H and is incremented 5 times in the loop.

The content of memory location 20H is 0, as we it is successively decremented by the `DJNZ` instruction until it reaches 0.

The contents of memory locations 80H-85H are unchanged, as we attempt to change the contents with the instruction `MOV @R0, A`, using indirect addressing, but these

locations are only accessible by direct addressing (see memory map in 8051 Programmer's Guide data sheet).

b) Find the number of bytes in code memory this instruction sequence occupies.

Instruction	Bytes	Machine Cycles	Opcode	Location (in hex)
start: MOV 2AH, #5	3	2	75 2A 05	0000
MOV R0, #80H	2	1	78 80	0003
CLR A	1	1	E4	0005
loop: MOV @R0, A	1	1	F6	0006
INC R0	1	1	08	0007
DJNZ 2AH,loop	3	2	D5 2A FB	0008
stop: SJMP stop	2	2	80 FE	000B

Total bytes in code memory = 3+2+1+1+1+3+2 = 13 bytes

c) Find the number of instruction cycles, machine cycles, clock cycles and total amount of time this code takes to complete (assume it's complete when it hits JMP stop). Assume the clock is running at 12 MHz.

Total machine cycles = (2+1+1) + 5*(1+1+2) +2 = 26 machine cycles

Each machine cycle is 12 clock cycles => 12*26 = 312 clock cycles

Each instruction takes one instruction cycle => (1+1+1) + 5*(1+1+1) + 1 = 19 instruction cycles.

Clock frequency = 12MHz => Clock period = 1 clock cycle = 83.33 ns

The code takes 312*83.33 = 26 us to execute.

d) Explain why this code does not change the value of special function registers P0, SP, DPL, etc., located at internal memory locations 80H, 81H, etc.

According to the 8051 data sheet, memory locations 80H through FFH can be accessed only by direct addressing. The code above attempts to modify these locations through indirect addressing.

e) Write the machine code and corresponding code memory locations for the above code segment (use a table). Assume that code memory starts at 0000H.

5. Problem 30 in Chapter 3 of ISM. Assume a 12 MHz clock.

Write a program to generate a 4 us active-high pulse on P1.7 every 200 us.

```
LOOP:     SETB P1.7       ;low  (1 us)
          NOP             ;high (1 us) begin pulse
          NOP             ;high (1 us)
          NOP             ;high (1 us)
          CLR P1.7        ;high (1 us) end pulse
          MOV R7,#96      ;low  (1 us)
WAIT:     DJNZ R7,WAIT    ;low  (96x2 = 192 us)
          SJMP LOOP       ;low  (2 us)
```

Generating a 4 us pulse every 200 us requires 196 ms low-time between pulses. The solution above uses a software loop to achieve the timing delay. The loop is a single 2 us instruction (`DJNZ`) that branches to itself. The loop is only 192 us because an addition 4 us is consumed by other instructions.

The 4 ms active-high pulse begins at the end of the `SETB P1.7` instruction and finishes at the end of the `CLR P1.7` instruction. Three `NOP` instructions are needed to stretch the high` time to 4us.