DEBRE BERHAN UNIVERSTY

COLLAGE OF COMPUTING

DEPARTMENT OF SOFTWARE ENGINEERING

NAME: YISHAQ DAMTEW

ID: DBU1601755

Submitted to: Ms. Rahel A

submitted date: 18/09/2017 e,c

QUESTION

1. An 8 bit register contain binary value 10101011. What is the register value after following shift microoperations.

A..Circular shift left	D.Logical shift right
B.Circular shift right	E.Arithmetic shift left
C.Logical shift left	F.Arithmetic shift right

Answer

(Initial = `10101011`)

A. Circular shift left: Rotate all bits left, wrapping the MSB around to LSB. For 10101011, the leftmost `1` goes to the rightmost position.

➔ Result = **`01010111`**.

B. Circular shift right: Rotate all bits right, wrapping the LSB to MSB. For 10101011, rightmost `1` becomes MSB.

➔ Result = **`11010101`**.

C. Logical shift left: Shift left, drop MSB, fill LSB with 0. 10101011 → drop leftmost `1`, append `0`:

➔ Result = **`01010110`**. (Logical shifts insert zeros.)

D. Logical shift right: Shift right, drop LSB, fill MSB with 0. 10101011 → drop rightmost `1`, prepend `0`:

➔ Result = **`01010101`**.

E. Arithmetic shift left: Same bit-shifting as logical left (fill 0), so **`01010110`** (identical to logical left for two's-complement). But it checks if thre is an overflow.

➔ Result = **`01010110`**.

F. Arithmetic shift right: Shift right preserving sign (MSB). Original MSB=1, so prepend `1` after dropping LSB. 10101011 →

➔ Result = **`11010101`** (MSB replicated).

•

QUESTION

2. Register A holds the binary information 11011001. Determine the B operand and the logic microoperation to be performed between A and B in order to change the value of A to:)

A. 01101101 B. 11111101

ANSWER

A = 11011001

We want to find a second operand B and a bitwise operation

a. Target 01101101: Here, only the XOR operation yields a unique solution. Compute `B = A XOR target` bitwise:

- A = 11011001
- B = 10110100 (calculated such that A⊕B = 01101101)

Thus use the **XOR** microoperation: `A ⊕ 10110100 = 01101101`. XOR flips exactly the bits needed (for each bit, 1⊕1=0, 0⊕1=1, etc.). (Logic microoperations act bitwise on registers.)

b. Target 11111101: Again choose XOR for a clear solution. Compute `B = A XOR target`:

- A = 11011001
- B = 00100100 (so that A⊕B = 11111101)

Use **XOR**: `11011001 ⊕ 00100100 = 11111101`. (Other ops like OR have many solutions, but XOR gives one direct answer.) in this case OR and XOR operations give us the same result.

In summary, for both cases the required micro-operation is **XOR**. In the first case `B=10110100`, and in the second `B=00100100`. Logic microoperations (AND/OR/XOR) operate bit by bit on register contents. XOR is often used to flip selected bits, which matches the needed transformations here.

QUESTION

3. List the main functions of the control unit. What are the advantages and disadvantages of micro programmed control compared with a hardwired control unit implementation?

ANSWER

Main functions of the Control Unit (CU): The CU orchestrates the CPU's activity. Its functions include:

- ✓ **Instruction fetching:** Retrieve the next instruction from memory (via the program counter).
- ✓ **Decoding:** Interpret the fetched instruction to determine required operation.
- ✓ **Control-signal generation:** Generate timing and control signals to direct the ALU, registers, memory, and I/O devices.
- ✓ **Data flow management:** Control movement of data between registers, ALU, and memory, and manage the instruction execution cycle.
- ✓ **Sequencing and flow control:** Step through the fetch–decode–execute cycle, manage conditional branches, interrupts, etc., changing control steps as needed.

In short, the CU "is the CPU's central nervous system": it uses the instruction bits (via a decoder) to produce the proper signals that direct the ALU and other units.

Hardwired vs. Microprogrammed Control:

- **Hardwired Control Unit:** Implements control signals with fixed logic circuits and finite-state hardware. It is very fast (no microinstruction fetch delays), making it suitable for simple or speed-critical CPUs. However, it is **inflexible**: changing or adding instructions requires redesigning hardware. It is complex and costly to modify. Hardwired control is generally used in RISC-like designs with few instructions.
- **Microprogrammed Control Unit:** Uses a stored microcode (in ROM) to sequence microinstructions that generate control signals. It is **slower** (each machine instruction may require multiple microinstructions), but much easier to update or extend (just change the microprogram). This flexibility means it can implement complex instruction sets or fix bugs in control logic by rewriting microcode. Microprogrammed control is simpler and cheaper in hardware, but it has higher cycle overhead.

Comparison (Advantages/Disadvantages): Hardwired control is faster and efficient for simple instruction sets, but difficult to modify or scale. Microprogrammed control, in contrast, is more adaptable and maintainable (good for complex/CISC designs) but incurs speed and storage costs. As one summary notes, hardwired designs "generate control signals via hardware" and are "faster" but "difficult to modify," whereas microprogrammed designs "generate control signals from stored micro instructions," making them "slower" but "easy to modify".

QUESTION

4. Discuss the different ways (modes of data transfer) by which data can be sent from the I/O device to the memory by the CPU?

ANSWER

Data transfer between an I/O device and memory can occur by different modes:

- **Programmed I/O (Polling):** The CPU repeatedly executes instructions to move data between the device and memory. For each word, the CPU issues an input instruction to read from the device and a store to memory, continuously checking (polling) if the device is ready. This keeps the CPU busy in a loop until transfer is complete. It does not allow CPU to do other work during transfer, making it inefficient for large or slow I/O (the CPU "needlessly keeps busy" with the I/O task).
- **Interrupt-driven I/O:** The device issues an interrupt when it is ready for transfer. The CPU can perform other tasks and only responds to the I/O when interrupted. Upon an interrupt request, the CPU saves context, executes an interrupt-service routine to handle the transfer (e.g. reading data into memory), then resumes. This reduces wasted CPU cycles compared to polling. However, each transfer still requires CPU intervention and context switch overhead.
- **Direct Memory Access (DMA):** A dedicated DMA controller handles the transfer of large blocks of data directly between the device and memory, without CPU intervention. The CPU initiates the DMA operation, then can do other work (or be idle) while the DMA handles the data over the bus. Once done, the DMA signals completion (often by interrupt). This achieves much higher throughput for bulk transfer because the CPU is largely bypassed. DMA is commonly used for high-speed peripherals (e.g. disk I/O).

These modes are often contrasted: programmed I/O ties up the CPU fully, interrupt I/O frees the CPU between interrupts (but CPU still services each byte), and DMA allows the CPU to be freed entirely during the bulk transfer.

QUESTION

5. CPU QUESTION

ANSWER

We have a CPU with 7 registers R1–R7 (4 bits each) and a 14-bit instruction format: 3-bit SELA, 3-bit SELB, 3-bit SELD, 5-bit OPR. SEL fields pick registers (e.g. `010` → R2) and OPR selects the ALU operation (see table below). Initial register values:

```
R1=1011, R2=1001, R3=0010, R4=0101, R5=1011, R6=1001, R7=0000
```

Execute the instructions in sequence, updating registers after each:

- **Instr 1: `01000110100010`**
  - ✓ SELA=`010` → R2, SELB=`001` → R1, SELD=`101` → R5, OPR=`00010`.
  - ✓ OPR=00010 = **ADD** (ALU computes A+B). So **R5 ← R2 + R1**.
  - ✓ R2=1001 (9) + R1=1011 (11) = 10100 (20); as 4-bit result (drop carry) **= 0100**.
  - ✓ **Decoder selected R5. Operation: `R5 = R2 + R1` (ADD). New R5 = 0100.**
- **Instr 2: `10111011101000`**
  - ✓ Updated R5 now 0100. SELA=`101` → R5 (0100), SELB=`110` → R6 (1001), SELD=`111` → R7, OPR=`01000`.
  - ✓ OPR=01000 = **AND** (bitwise AND). So **R7 ← R5 AND R6**.
  - ✓ R5(0100) & R6(1001) = 0000.
  - ✓ **Decoder selects R7. Operation: `R7 = R5 AND R6`. New R7 = 0000.**
- **Instr 3: `10110010101100`**
  - ✓ Updated R5 is 0001 (from previous XOR), but wait – we must update R5 from step 2: R5 remained 0100 (only R7 changed). So now R5=0100, R6=1001. SELA=`101` → R5 (0100), SELB=`100` → R4 (0101), SELD=`101` → R5, OPR=`01100`.
  - ✓ OPR=01100 = **XOR** (bitwise XOR). So **R5 ← R5 XOR R4**.
  - ✓ 0100 XOR 0101 = **0001**.
  - ✓ **Decoder selects R5. Operation: `R5 = R5 XOR R4`. New R5 = 0001.**
- **Instr 4: `10011001001010`**
  - ✓ Current R5=0001, R4=0101, R6=1001, R2=1001. SELA=`100` → R4 (0101), SELB=`110` → R6 (1001), SELD=`010` → R2, OPR=`01010`.
  - ✓ OPR=01010 = **OR** (bitwise OR). So **R2 ← R4 OR R6**.
  - ✓ 0101 OR 1001 = **1101**.
  - ✓ **Decoder selects R2. Operation: `R2 = R4 OR R6`. New R2 = 1101.**

The selector bits choose registers by binary number, and OPR codes match the ALU table (ADD=00010, AND=01000, OR=01010, XOR=01100). We applied each to the current register values and noted the updated target register.

QUESTION

6. Write full assembly program to print a string 'My First Assembly Language Program' on the screen.

ANSWER

Print "My First Assembly Language Program":

```
; Program to display a string using DOS interrupt 21h function 09
.model small
.stack 100h
.data
    msg db 'My First Assembly Language Program$'
```

```
.code
main:
    mov ax, @data
    mov ds, ax
    mov dx, offset msg
    mov ah, 09h
    int 21h              '$'):contentReference[oaicite:49]{index=49}
    mov ah, 4Ch
    int 21h
end main
```

Explanation: The string is defined in `.data` ending with a `$`. We load DS and DX so that INT 21h/AH=09 prints the string. After printing, AH=4Ch exits.

QUESTION

7.Write a complete program to display the letter ‘a’ on the screen.

ANSWER

Display the character ‘a’:

```
; Program to display character 'a' using DOS interrupt 21h function 02
.model small
.stack 100h
.data
    char db 'a'
.code
main:
    mov ax, @data
    mov ds, ax          ; Initialize data segment
    mov dl, char        ; DL = ASCII of 'a'
    mov ah, 02h         ; DOS print character function
    int 21h             ; Print DL:contentReference[oaicite:51]{index=51}
    mov ah, 4Ch
    int 21h
end main
```

Explanation: We load the ASCII code of `'a'` into DL and call INT 21h/AH=02 to print that character. This writes `'a'` to screen.

QUESTION

8. Write a program to load character ‘?’ into register ax and display the same on the screen.

ANSWER

Load '?' into AX and display it:

```
; Program to load '?' into AX and display it
.model small
.stack 100h
.code
main:
    mov ax, 003Fh      ; '?' = 3Fh in ASCII; loaded into AL (AX=0003Fh)
    mov dl, al        ; Move character '?' into DL
    mov ah, 02h       ; Print character in DL
    int 21h           ; Display '?':contentReference[oaicite:53]{index=53}
    mov ah, 4Ch
    int 21h
end main
```

Explanation: We load AX with the ASCII code of ? (0x3F), then move AL into DL.
INT 21h/AH=02 prints the character in DL. This shows ? on screen.

QUESTION

9. Write full assembly Program to add consecutive 10 numbers.

ANSWER

Add 10 consecutive numbers (1 through 10) and display sum:

```
; Program to sum numbers 1 to 10 and print the result (55)
.model small
.stack 100h
.code
main:
    mov ax, @data
    mov cx, 10
    mov al, 0
loop1:
    add al, cl
    loop loop1
    mov ah, 0
    mov bl, 10
    div bl            ; AL=5, AH=5
    add al, '0'       ; AL = '5'
    mov dl, al
    mov ah, 02h
    int 21h
    add ah, '0'
    mov dl, ah
    mov ah, 02h
    int 21h
```

```
    mov ah, 4Ch
    int 21h
end main
```

Explanation: We loop CX=10 down to 1, adding each value to AL. The final sum in AL is 55 (decimal). We then divide by 10 to get tens and units (result=5 and remainder=5), convert to ASCII by adding `30h`, and print each digit with INT 21h/AH=02.

QUESTION

10. Write an interactive ALP to read a byte from the keyboard and display it on the screen.

ANSWER

Interactive input and display of a byte:

```
; Program to read a character from keyboard and display it
.model small
.stack 100h
.code
main:
    mov ah, 01h
    int 21h
    mov dl, al
    mov ah, 02h
    int 21h           ; Display the same
character:contentReference[oaicite:56]{index=56}:contentReference[oaicite:57]
{index=57}
    mov ah, 4Ch
    int 21h
end main
```

Explanation: INT 21h/AH=01h reads a keystroke into AL (and echoes it). We then move AL into DL and call INT 21h/AH=02h to explicitly display it again. The result is an interactive read-then-echo of the input byte.

Each program uses standard DOS interrupts for I/O. For instance, function 09h prints a `$`-terminated string and function 02h prints the character in DL. These examples demonstrate basic ALP concepts (data segments, registers, loops, and interrupt calls) to accomplish the given tasks.

Sources: Definitions and examples of shifts, control unit roles, control unit design trade-offs, I/O transfer modes, register/ALU microoperations, and DOS interrupt usage were used to support this solution.