CONDITIONAL PROCESSING

MODULE OUTLINE

1. BOOLEAN AND COMPARISON INSTRUCTIONS
2. CONDITIONAL JUMPS
3. CONDITIONAL LOOP INSTRUCTIONS
4. CONDITIONAL STRUCTURES
5. APPLICATION: FINITE-STATE MACHINES
6. CONDITIONAL CONTROL FLOW DIRECTIVES

8.1. BOOLEAN AND COMPARISON INSTRUCTIONS - 1

Operation	Description
AND	Boolean AND operation between a source operand and a destination operand.
OR	Boolean OR operation between a source operand and a destination operand.
XOR	Boolean exclusive-OR operation between a source operand and a destination operand.
NOT	Boolean NOT operation on a destination operand.
TEST	Implied boolean AND operation between a source and destination operand, setting the CPU flags appropriately.
BT, BTC, BTR, BTS	Copy bit *n* from the source operand to the Carry flag and complement/reset/set the same bit in the destination operand (covered in Section 6.3.5).

Table 8.1.1: Selected Boolean Instructions

8.1. BOOLEAN AND COMPARISON INSTRUCTIONS - 2

8.1.1. CPU FLAGS

- **Boolean instructions affect the Zero, Carry, Sign, Overflow, and Parity flags**

8.1.2. AND INSTRUCTION

- **General Syntax**

```
AND destination,source
```

- **Actual Syntax**

```
AND reg,reg
AND reg,mem
AND reg,imm
AND mem,reg
AND mem,imm
```

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

8.1. BOOLEAN AND COMPARISON INSTRUCTIONS - 3

8.1.2. AND INSTRUCTION

- **BIT MASKING**

```
and AL,11110110b          ;clear bits 0 and 3
```

- **EXAMPLE: AL contains 10101110**

```
mov al,10101110b
and al,11110110b          ;result in AL = 10100110
```

8.1. BOOLEAN AND COMPARISON INSTRUCTIONS - 4

8.1.2. AND INSTRUCTION

- **CONVERTING CHARACTERS TO UPPER CASE**

```
0 1 10 0 0 0 1 = 61h ('a')
0 1 00 0 0 0 1 = 41h ('A')
```

- **EXAMPLE:**

```
.data
array DB 50 "This Sentence is in Mixed Case",0
arrLen EQU $-array
.text
     mov ecx,arrLen
     mov esi,[array]
L1: and DB [esi],11011111b  ;clear bit 5
    inc esi
    loop L1
```

8.1. BOOLEAN AND COMPARISON INSTRUCTIONS - 5

8.1.3. OR INSTRUCTION

- **General Syntax**

```
OR destination,source
```

- **Actual Syntax**

```
OR reg,reg
OR reg,mem
OR reg,imm
OR mem,reg
OR mem,imm
```

x	y	x ∨ y
0	0	0
0	1	1
1	0	1
1	1	1

8.1. BOOLEAN AND COMPARISON INSTRUCTIONS - 6

8.1.3. OR INSTRUCTION

- **BIT SETTING**

```
or AL,00000100b          ;set bit 2
```

- **EXAMPLE: AL contains 11100011**

```
mov AL,11100011b
or AL,00000100b          ;result in AL = 11100111
```

Zero Flag	Sign Flag	Value in AL Is . . .
Clear	Clear	Greater than zero
Set	Clear	Equal to zero
Clear	Set	Less than zero

8.1. BOOLEAN AND COMPARISON INSTRUCTIONS - 7

8.1.4. BIT-MAPPED SETS

- **EXAMPLE**

```
SetX = 10000000 00000000 00000000 00000111
mov eax,SetX
and eax,10000b    ;does element[16] belong to SetX?
```

- **SET COMPLEMENT**

```
mov eax,SetX
not eax           ;complement of SetX
```

- **SET INTERSECTION**

```
SetX = 1000000 00000000 00000000 00000111
SetY = 1000001 01010000 00000111 01100011
mov EAX, SetX
and EAX, SetY
;SetX ∩ SetY = 1000000 00000000 00000000 00000011
```

- **SET UNION**

```
mov EAX, SetX
or EAX, SetY
;SetX ∪ SetY = 1000001 01010000 00000111 01100111
```

8.1. BOOLEAN AND COMPARISON INSTRUCTIONS - 8

8.1.5. XOR INSTRUCTION

- **General Syntax**

```
XOR destination,source
```

- **Actual Syntax**

```
XOR reg,reg
XOR reg,mem
XOR reg,imm
XOR mem,reg
XOR mem,imm
```

x	y	x ⊕ y
0	0	0
0	1	1
1	0	1
1	1	0

8.1. BOOLEAN AND COMPARISON INSTRUCTIONS - 9

8.1.6. NOT INSTRUCTION

- **General Syntax**

```
NOT destination
```

- **Actual Syntax**

```
NOT reg
NOT mem
```

- **EXAMPLE**

```
mov AL,11110000b
not AL                          ;AL = 00001111b
```

8.1. BOOLEAN AND COMPARISON INSTRUCTIONS - 10

8.1.7. TEST INSTRUCTION

- **General Syntax**

```
test source, bitMask

test al,00001001b
```

- **Testing Multiple Bits**

```
test al,00001001b     ;test bits 0 and 3
```

8.1. BOOLEAN AND COMPARISON INSTRUCTIONS - 11

8.1.8. CMP INSTRUCTION

- **Here is problem (expressed in pseudocode)**

```
if A > B then ...
while X > 0 and X < 200 ...
if check_for_error( N ) = true then
```

- **Examples**

```
mov ax,5
cmp ax,10          ;ZF = 0 and CF = 1
mov ax,1000
mov cx,1000
cmp cx,ax          ;ZF = 1 and CF = 0
mov si,105
cmp si,0           ;ZF = 0 and CF = 0
```

8.1. BOOLEAN AND COMPARISON INSTRUCTIONS - 12

8.1.8. SETTING AND CLEARING CPU FLAGS

- **Here is problem (expressed in pseudocode)**

```
if A > B then ...
while X > 0 and X < 200 ...
if check_for_error( N ) = true then
```

- **Examples**

```
mov ax,5
cmp ax,10          ;ZF = 0 and CF = 1
mov ax,1000
mov cx,1000
cmp cx,ax          ;ZF = 1 and CF = 0
mov si,105
cmp si,0           ;ZF = 0 and CF = 0
```

8.1. BOOLEAN AND COMPARISON INSTRUCTIONS - 13

8.1.8. CMP INSTRUCTION

- **Examples**

```
test AL,0          ;set Zero flag
and AL,0           ;set Zero flag
or AL,1            ;clear Zero flag
or AL,80h          ;set Sign flag
and AL,7Fh         ;clear Sign flag
stc                ;set Carry flag
clc                ;clear Carry flag
mov AL,7Fh         ;AL = +127
inc AL             ;AL = 80h (-128), OF=1
or eax,0           ;clear Overflow flag
```

8.2. CONDITIONAL JUMPS - 1

8.2.1. CONDITIONAL STRUCTURES

- **Example 1 (JZ – jump if Zero)**

```
        cmp EAX,0
        jz L1                       ;jump if ZF = 1
        .
        .
L1:
```

- **Example 2 (JNZ – jump if Not Zero)**

```
        and DL,10110000b
        jnz L2                      ;jump if ZF = 0
        .
        .
L2:
```

8.2. CONDITIONAL JUMPS - 2

8.2.2. JCOND INSTRUCTION

- **General Syntax**

```
Jcond destination
```

jc	Jump if carry (Carry flag set)
jnc	Jump if not carry (Carry flag clear)
jz	Jump if zero (Zero flag set)
jnz	Jump if not zero (Zero flag clear)

Table 8.2.2.1: Different types of Jcond

8.2. CONDITIONAL JUMPS - 3

8.2.2. JCOND INSTRUCTION

- **Using CMP Instruction**
  - **Checking for Equality**

```
cmp eax,5
je L1                          ;jump if equal
```

  - **Checking for Inequality**

```
mov ax,5
cmp ax,6
jl L1                          ;jump if less

mov ax,5
cmp ax,4
jg L1                          ;jump if greater
```

8.2. CONDITIONAL JUMPS - 4

8.2.3. TYPES OF CONDITIONAL JUMPS INSTRUCTIONS

- **Jumps based on specific flag values**
- **Jumps based on comparison of operands or the value of (E)CX**
- **Jumps based on comparisons of unsigned operands**
- **Jumps based on comparisons of signed operands**

8.2. CONDITIONAL JUMPS - 5

8.2.3. TYPES OF CONDITIONAL JUMPS INSTRUCTIONS

Mnemonic	Description	Flags / Registers
JZ	Jump if zero	ZF = 1
JNZ	Jump if not zero	ZF = 0
JC	Jump if carry	CF = 1
JNC	Jump if not carry	CF = 0
JO	Jump if overflow	OF = 1
JNO	Jump if not overflow	OF = 0
JS	Jump if signed	SF = 1
JNS	Jump if not signed	SF = 0
JP	Jump if parity (even)	PF = 1
JNP	Jump if not parity (odd)	PF = 0

Table 8.2.3.1: Jumps Based on Specific Flag Values

8.2. CONDITIONAL JUMPS - 6

8.2.3. TYPES OF CONDITIONAL JUMPS INSTRUCTIONS

- **Equality Comparisons**

```
CMP leftOp, rightOp
```

Mnemonic	Description
JE	Jump if equal (*leftOp* = *rightOp*)
JNE	Jump if not equal (*leftOp* ≠ *rightOp*)
JCXZ	Jump if CX = 0
JECXZ	Jump if ECX = 0

Table 8.2.3.2: Jumps Based on Equality

8.2. CONDITIONAL JUMPS - 7

8.2.3. TYPES OF CONDITIONAL JUMPS INSTRUCTIONS

- **Example 1**

```
mov edx,0A523h
cmp edx,0A523h
jne L5                  ;jump not taken
je L1                   ;jump is taken
```

Table 8.2.3.2: Jumps Based on Equality

8.2. CONDITIONAL JUMPS - 8

8.2.3. TYPES OF CONDITIONAL JUMPS INSTRUCTIONS

Mnemonic	Description
JA	Jump if above (if *leftOp* > *rightOp*)
JNBE	Jump if not below or equal (same as JA)
JAE	Jump if above or equal (if *leftOp* ≥ *rightOp*)
JNB	Jump if not below (same as JAE)
JB	Jump if below (if *leftOp* < *rightOp*)
JNAE	Jump if not above or equal (same as JB)
JBE	Jump if below or equal (if *leftOp* ≤ *rightOp*)
JNA	Jump if not above (same as JBE)

Table 8.2.3.3: Jumps Based on Unsigned Comparisons

8.2. CONDITIONAL JUMPS - 9

8.2.3. TYPES OF CONDITIONAL JUMPS INSTRUCTIONS

Mnemonic	Description
JG	Jump if greater (if *leftOp* > *rightOp*)
JNLE	Jump if not less than or equal (same as JG)
JGE	Jump if greater than or equal (if *leftOp* ≥ *rightOp*)
JNL	Jump if not less (same as JGE)
JL	Jump if less (if *leftOp* < *rightOp*)
JNGE	Jump if not greater than or equal (same as JL)
JLE	Jump if less than or equal (if *leftOp* ≤ *rightOp*)
JNG	Jump if not greater (same as JLE)

Table 8.2.3.4: Jumps Based on Signed Comparisons

8.2. CONDITIONAL JUMPS - 10

8.2.3. TYPES OF CONDITIONAL JUMPS INSTRUCTIONS

- **Example 1**

```
mov edx,-1
cmp edx,0
jnl L5        ;jump not taken (-1 >= 0 is false)
jnle L5       ;jump not taken (-1 > 0 is false)
jl L1         ;jump is taken (-1 < 0 is true)
```

- **Example 2**

```
mov bx,+32
mp bx,-35
jng L5        ;jump not taken (+32 <= -35 is false)
jnge L5       ;jump not taken (+32 < -35 is false)
jge L1        ;jump is taken (+32 >= -35 is true)
```

8.2. CONDITIONAL JUMPS - 11

8.2.4. CONDITIONAL JUMP APPLICATIONS

- **Testing Status Bits**

```
mov al,status
test al,00100000b          ;test bit 5
jnz EquipOffline

mov al,status
test al,00010011b          ;test bits 0,1,4
jnz InputDataByte

mov al,status
and al,10001100b           ;mask bits 2,3,7
cmp al,10001100b           ;all bits set?
je ResetMachine            ;yes: jump to label
```

8.2. CONDITIONAL JUMPS - 12

8.2.4. CONDITIONAL JUMP APPLICATIONS

- **Larger of Two Integers**

```
        mov edx,eax                 ;assume EAX is larger
        cmp eax,ebx                 ;if EAX is >= EBX then
        jae L1              ;jump to L1
        mov edx,ebx                 ;else move EBX to EDX
L1:                         ;EDX contains the larger integer
```

- **Smallest of Three Integers**

```
         .bss
        V1 DW ?
        V2 DW ?
        V3 DW ?
        .text
        mov ax,V1           ;assume V1 is smallest
        cmp ax,V2           ;if AX <= V2 then
        jbe L1              ;jump to L1
        mov ax,V2           ;else move V2 to AX
L1:     cmp ax,V3           ;if AX <= V3 then
        jbe L2              ;jump to L2
        mov ax,V3           ;else move V3 to AX
L2:
```

8.2. CONDITIONAL JUMPS - 13

8.2.4. CONDITIONAL JUMP APPLICATIONS

- **Loop Until Key Pressed**

```
.bss
char DB ?
.text
L1:     mov eax,10      ;create 10ms delay
        call Delay
        call ReadKey    ;check for key
        jz L1           ;repeat if no key
        mov char,AL     ;save the character
```

8.3. CONDITIONAL LOOP INSTRUCTIONS - 1

8.3.1. LOOPZ AND LOOPE INSTRUCTIONS

- **General Syntax**

```
LOOPZ destination
LOOPE destination
```

They perform the following tasks:

```
ECX = ECX – 1
if ECX = 0 and ZF = 1, jump to destination
```

LOOPZ and LOOPE don’t affect any of the status flags

8.3. CONDITIONAL LOOP INSTRUCTIONS - 2

8.3.1. LOOPNZ AND LOOPNE INSTRUCTIONS

- **General Syntax**

```
LOOPNZ destination
LOOPNE destination
```

They perform the following tasks:

```
ECX = ECX – 1
if ECX > 0 and ZF = 0, jump to destination
```

LOOPNZ and LOOPNE don’t affect any of the status flags

8.3. CONDITIONAL LOOP INSTRUCTIONS - 3

8.3.1. LOOPNZ AND LOOPNE INSTRUCTIONS

- **Example**

```
.data
array DW -3,-6,-1,-10,10,30,40,4
sentinel DW 0
arrLen EQU $-array
.text
     mov esi,[array]
     mov ecx,arrLen
L1:  test DW [esi],8000h          ;test sign bit
     pushfd                       ;push flags on stack
     add esi,DW array             ;move to next position
     popfd                        ;pop flags from stack
     loopnz L1                    ;continue loop
     jnz quit                     ;none found
     sub esi,DW array             ;ESI points to value
quit:
```

WARNING: These instructions are slow on modern CPUs!!!

8.4. CONDITIONAL STRUCTURES - 1

8.4.1. BLOCK-STRUCTURED IF STATEMENTS

- **General Syntax (below is pseudocode)**

```
if( boolean-expression)
   statement-list-1
else
   statement-list-2
```

- **Example**

```
mov eax,op1
cmp eax,op2                ;op1 == op2?
jne L1                     ;no: skip next
mov X,1                    ;yes: assign X and Y
mov Y,2
L1:
```

8.4. CONDITIONAL STRUCTURES - 2

8.4.1. BLOCK-STRUCTURED IF STATEMENTS

- **Following Situation (below is pseudocode)**

```
if op1 > op2 then
    call Routine1
else
    call Routine2
end if
```

- **Example**

```
    mov eax,op1                ;move op1 to a register
    cmp eax,op2                ;op1 > op2?
    jg A1                      ;yes: call Routine1
    call Routine2              ;no: call Routine2
    jmp A2                     ;exit the IF statement
A1: call Routine1
A2:
```

8.4. CONDITIONAL STRUCTURES - 3

8.4.1. BLOCK-STRUCTURED IF STATEMENTS

- **White Box Testing (below is pseudocode)**

```
if op1 == op2 then
  if X > Y then
     call Routine1
  else
     call Routine2
  end if
else
  call Routine3
end if
```

- **Example**

```
1:        mov eax,op1
2:        cmp eax,op2                ;op1 == op2?
3:        jne L2                     ;no: call Routine3
;process the inner IF-ELSE statement.
4:        mov eax,X
5:        cmp eax,Y                          ;X > Y?
6:        jg L1                      ;yes: call Routine1
7:        call Routine2              ;no: call Routine2
8:        jmp L3                     ;and exit
9:   L1:  call Routine1              ;call Routine1
10:       jmp L3                     ;and exit
11:  L2:  call Routine3
12:  L3:
```

8.4. CONDITIONAL STRUCTURES - 4

8.4.2. COMPOUND EXPRESSIONS

- **Logical AND Operator (below is pseudocode)**

```
if (a1 > b1) AND (b1 > c1) then
   X = 1
end if
```

- **Short-Circuit Evaluation**

```
      cmp al,bl                    ;first expression...
      ja L1
      jmp next
L1: cmp bl,cl                      ;second expression...
      ja L2
      jmp next
L2: mov X,1                        ;both true: set X to 1
next:
```

8.4. CONDITIONAL STRUCTURES - 5

8.4.2. COMPOUND EXPRESSIONS

- **Short-Circuit Evaluation - contd**

```
        cmp al,bl       ;first expression...
        jbe next        ;quit if false
        cmp bl,cl       ;second expression
        jbe next        ;quit if false
        mov X,1         ;both are true
    next:
```

8.4. CONDITIONAL STRUCTURES - 6

8.4.2. COMPOUND EXPRESSIONS

- **Logical OR Operator (below is pseudocode)**

```
if (a1 > b1) OR (b1 > c1) then
   X = 1
end if
```

- **Example**

```
        cmp al,bl     ;1: compare AL to BL
        ja L1         ;if true, skip second expression
        cmp bl,cl     ;2: compare BL to CL
        jbe next      ;false: skip next statement
L1:     mov X,1       ;true: set X = 1
next:
```

8.4. CONDITIONAL STRUCTURES - 7

8.4.2. WHILE LOOPS

- **C++ Syntax**

```
while( val1 < val2 )
{
    val1++;
    val2--;
}
```

- **Assembly Programming:**

```
        mov eax,val1        ;copy variable to EAX
beginwhile:
        cmp eax,val2        ;if not (val1 < val2)
        jnl endwhile        ;exit the loop
        inc eax             ;val1++;
        dec val2            ;val2--;
        jmp beginwhile      ;repeat the loop
endwhile:
        mov val1,eax        ;save new value for val1
```

8.5. APPLICATION: FINITE STATE MACHINES - 1

8.5.1. INTRODUCTION

- **A finite-state machine(FSM) is a machine or program that changes state based on some input**

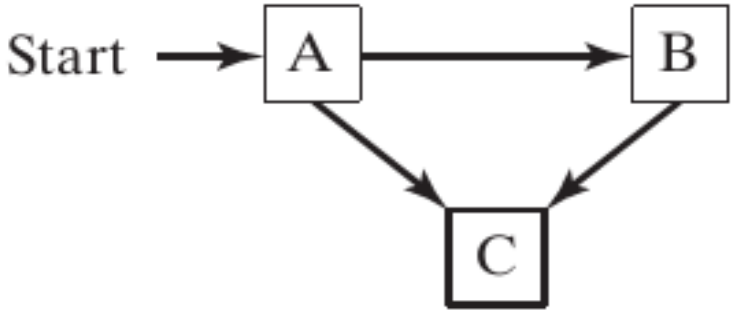

Figure 8.5.1.1: Simple Finite-State Machine

8.5. APPLICATION: FINITE STATE MACHINES - 2

8.5.2. VALIDATING AN INPUT STRING

- **Character String Example: Let's check the validity of an input string according to the following two rules:**
  - **The string must begin with the letter 'x' and end with the letter 'z.'**
  - **Between the first and last characters, there can be zero or more letters within the range {'a'...'y'}.**

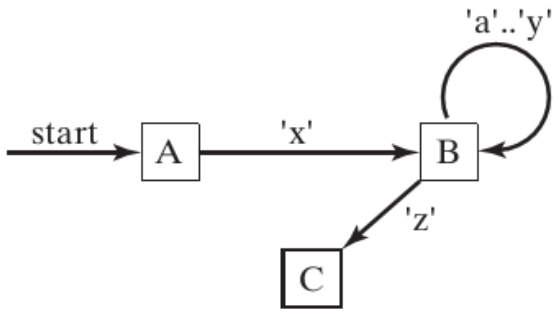

Figure 8.5.2.1: FSM for String

8.5. APPLICATION: FINITE STATE MACHINES - 3

8.5.3. VALIDATING A SIGNED INTEGER

- **Signed Integer Example: Input consists of an optional leading sign followed by a sequence of digits.**

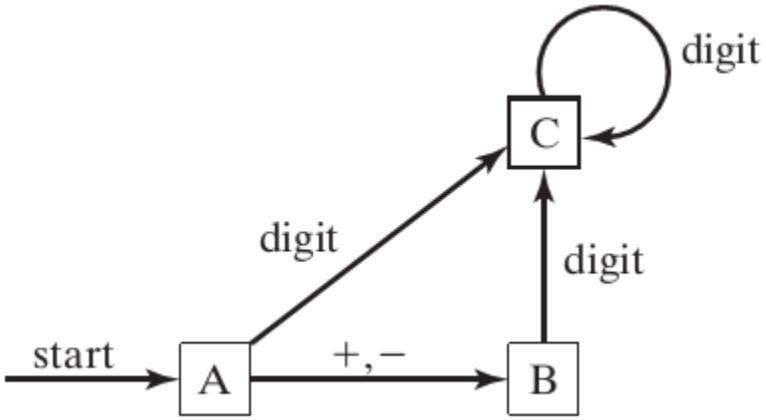

Figure 8.5.3.1: Signed Decimal Integer FSM