Branching II

Flags, Jump Types, Nested Loops

FLAGs associated with jumps

Unsigned Comparisons

JB	CF = 1 (borrow)
JAE	CF = 0 (no borrow)
JA	CF = 0 AND ZF = 0
JBE	CF = 1 OR ZF = 1

For example,

```
mov ah, 0x75
sub ah, 0xD2
jb target
```

75-D2 = A3 with a borrow

FLAGs associated with jumps

Signed Comparisons

JL	SF ≠ OF
JGE	SF = OF
JG	(SF = OF) AND ZF = 0
JLE	(SF ≠ OF) OR ZF = 1

For example,

```
mov ah, 26
sub ah, 50
jl target
```

26-50 = -24, or E8h, or 1110 1000
SF = 1 (negative value)
OF = 0 (no overflow)

FLAGs associated with jumps

Signed Comparisons

JL	SF ≠ OF
JGE	SF = OF
JG	(SF = OF) AND ZF = 0
JLE	(SF ≠ OF) OR ZF = 1

For example,

```
mov ah, 110
sub ah, -90
jl target
```

110-(-90) = 200 but it overflows
AH becomes C8h, or 1100 1000
SF = 1 (MSB is 1)
OF = 1 (overflow occurred)

Types of Jumps

Encoding of Jump Instructions

```
; program to add a list of numbers in memory
[org 0x100]
jmp start

data:   dw  5, 9, 13, 20, 23, 7, 11, 3, 87, 20
sum:    dw  0

start:
        mov ax, 0       ; accumulator
        mov bx, 0       ; index

while:  add ax, [bx+data]
        add bx, 2
        cmp bx, 20
        jne while

        mov [sum], ax

        mov ax, 0x4c00
        int 0x21
```

Note how **jmp start** instruction is translated to machine code.
Opcode: **E9**
Operand: **00 16 (displacement)**

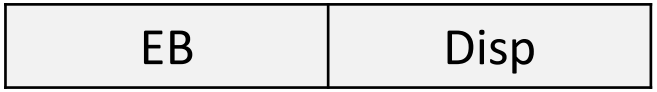
```
AX 0000    SI 0000    CS 20BB    IP 0100
BX 0000    DI 0000    DS 20BB
CX 0035    BP 0000    ES 20BB    HS 20BB
DX 0000    SP FFFE    SS 20BB    FS 20BB

CMD >

0100 E91600          JMP     0119
0103 050009          ADD     AX,0900
0106 000D            ADD     [DI],CL
0108 0014            ADD     [SI],DL
010A 0017            ADD     [BX],DL
010C 0007            ADD     [BX],AL
010E 000B            ADD     [BP+DI],CL
0110 0003            ADD     [BP+DI],AL
```

Encoding of Jump Instructions

- Short Jump



EB, E9, EA are all opcodes of JMP instruction

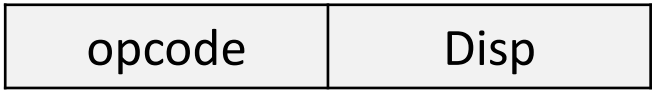
- Near Jump

- Far Jump

Short Jump

- Contains a one-byte position-relative offset (displacement)
- Offset is added to IP as a signed number
- Can go +127 bytes ahead in code and -128 bytes backwards

- Conditional jumps can only by short



Near Jump

- Contains a relative displacement of 16 bits
- Can jump anywhere within the code segment (16-bit signed number can be up to +/- 32K)
- A large relative jump causes a wraparound within the segment

Far Jump

- Not position-relative, but absolute target address is specified.
- Can jump to anywhere in memory
- Only three instructions have a far jump form: JMP, CALL, RET

opcode	IP Low	IP High	CS Low	CS High

Compound Boolean Expressions

Logical AND

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

```
     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:
```

First attempt: short-circuit evaluation

Logical AND

```
if (al > bl) AND (bl > cl)
     X = 1
end if
```

```
     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:
```

Another try:
smaller code

Logical OR

```
if (al > bl) OR (bl > cl)
    X = 1
```

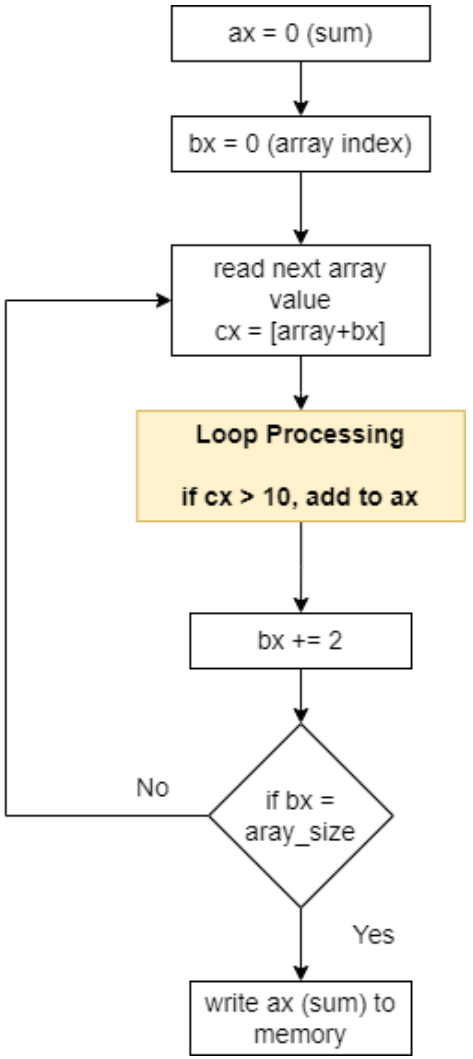
```
     cmp al,bl ; 1: compare AL to BL
     ja L1     ; if true, skip second expression
     cmp bl,cl ; 2: compare BL to CL
     jbe next  ; false: skip next statement
L1:  mov X,1   ; true: set X = 1
next:
```

Again, short-circuiting preferred

Nested Loops and Conditions

Add Selected Numbers from List

- Given a list of numbers, add only those values that are more than 10



Add Selected Numbers from List

```
; Add up only those numbers is a list that are bigger than 10
[org 100h]
jmp start

data:  dw  5, 9, 13, 20, 23, 7, 11, 3, 87, 20
sum:   dw  0

start:
         mov ax, 0      ; accumulator
         mov bx, 0      ; index

repeat:  mov dx, [bx+data]  ; get next piece of data
         cmp dx, 10
         jbe advance    ; if value > 10, then execute next line, otherwise skip it
         add ax, dx
advance: add bx, 2
         cmp bx, 20     ; end of array reached?
         jne repeat

         mov [sum], ax

         mov ax, 0x4c00
         int 0x21
```

Bubble Sort Example

State of Data	Swap Done	Swap Flag
Pass 1		Off
60 55 45 58	Yes	On
55 60 45 58	Yes	On
55 45 60 58	Yes	On
Pass 2		Off
55 45 58 60	Yes	On
45 55 58 60	No	On
45 55 58 60	No	On

Bubble Sort Example

State of Data	Swap Done	Swap Flag
Pass 3		Off
45 55 58 60	No	Off
45 55 58 60	No	Off
45 55 58 60	No	Off

No more passes since swap flag is Off

Bubble Sort Example

Example 3.3

```
001     ; sorting a list of ten numbers using bubble sort
002     [org 0x0100]
003                     jmp  start
004
005     data:           dw   60, 55, 45, 50, 40, 35, 25, 30, 10, 0
006     swap:           db   0
007
008     start:          mov  bx, 0              ; initialize array index to zero
009                     mov  byte [swap], 0     ; rest swap flag to no swaps
010
011     loop1:          mov  ax, [data+bx]      ; load number in ax
012                     cmp  ax, [data+bx+2]    ; compare with next number
013                     jbe  noswap             ; no swap if already in order
014
015                     mov  dx, [data+bx+2]    ; load second element in dx
016                     mov  [data+bx+2], ax    ; store first number in second
017                     mov  [data+bx], dx      ; store second number in first
018                     mov  byte [swap], 1     ; flag that a swap has been done
019
020     noswap:         add  bx, 2              ; advance bx to next index
021                     cmp  bx, 18             ; are we at last index
022                     jne  loop1              ; if not compare next two
023
024                     cmp  byte [swap], 1     ; check if a swap has been done
025                     je   start              ; if yes make another pass
026
027                     mov  ax, 0x4c00         ; terminate program
028                     int  0x21
```

References

- BH 3.4 to 3.6
- KI 6.5.2