LOOPS LOOPS LOOPS

Presented by:
Jerry Auvagha - 102321
Ayan Keynan - 093268
Shannon Mujera - 101981
Cosmas Nyairo - 101193
Tiffany Tirop - 099365

Introduction

Loops are **control structures** that allow a block of code to be executed until a condition is met.

Important Concepts

Loop’s anatomy

- Labels
- Unconditional Control Instructions
- Conditional Control Instructions
- CMP and Loop Instruction
- Jump Out Of Range Error

Labels

A program label is the target, or a location to jump to, for control statements. For example, the start of a loop might be marked with a label such as “loopStart”. The code may be re-executed by jumping to the label.

Labels: naming convention

Generally, a label starts with a letter, followed by letters, numbers, or symbols (limited to “_”), terminated with a colon (“:”).

It is possible to start labels with non-letter characters (i.e., digits, “_”, “$”, “#”, “@”, “~” or “?”) . However, these typically convey special meaning and, in general, should not be used by programmers. Labels in yasm are case sensitive.

For example, loopStart: last: are valid labels. Program labels may be defined only once.

Unconditional Control Instructions: The JMP instruction

The unconditional instruction provides an unconditional jump to a specific location in the program denoted with a program label.

Transfer of control may be forward, to execute a new set of instructions or backward, to re-execute the same steps.

The JMP instruction performs an unconditional jump. Such an instruction transfers the flow of execution by changing the instruction pointer register.

Unconditional Control Instruction: Example

```
MOV   AX, 00    ; Initializing AX to 0
MOV   BX, 00    ; Initializing BX to 0
MOV   CX, 01    ; Initializing CX to 1
LE_LOOP:
ADD   AX, 01    ; Increment AX
ADD   BX, AX    ; Add AX to BX
SHL   CX, 1     ; shift left CX, this in turn
;doubles the CX value
JMP   LE_LOOP   ; repeats the statements
```

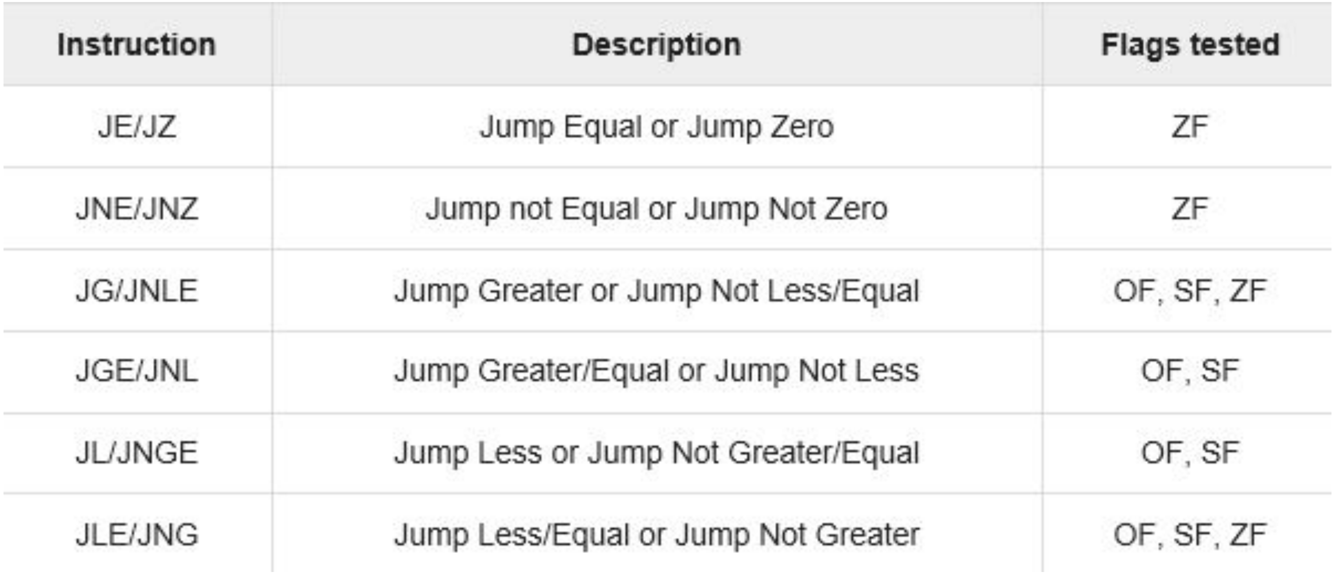
Conditional Control Instructions

Also known as **conditional jump instructions.**

If some specified condition is satisfied in conditional jump, the control flow is transferred to a target instruction.

List of Conditional Jumps: Arithmetic Operations

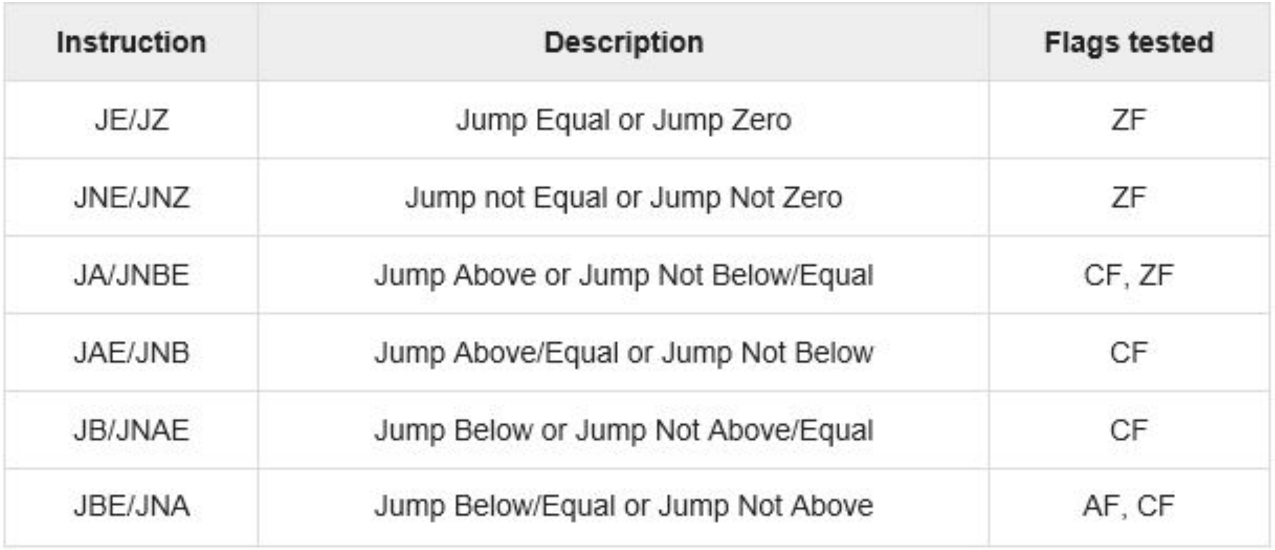
Following are the conditional jump instructions used on signed data used for arithmetic operations



Instruction	Description	Flags tested
JE/JZ	Jump Equal or Jump Zero	ZF
JNE/JNZ	Jump not Equal or Jump Not Zero	ZF
JG/JNLE	Jump Greater or Jump Not Less/Equal	OF, SF, ZF
JGE/JNL	Jump Greater/Equal or Jump Not Less	OF, SF
JL/JNGE	Jump Less or Jump Not Greater/Equal	OF, SF
JLE/JNG	Jump Less/Equal or Jump Not Greater	OF, SF, ZF

List of Conditional Jumps: Logical Operations

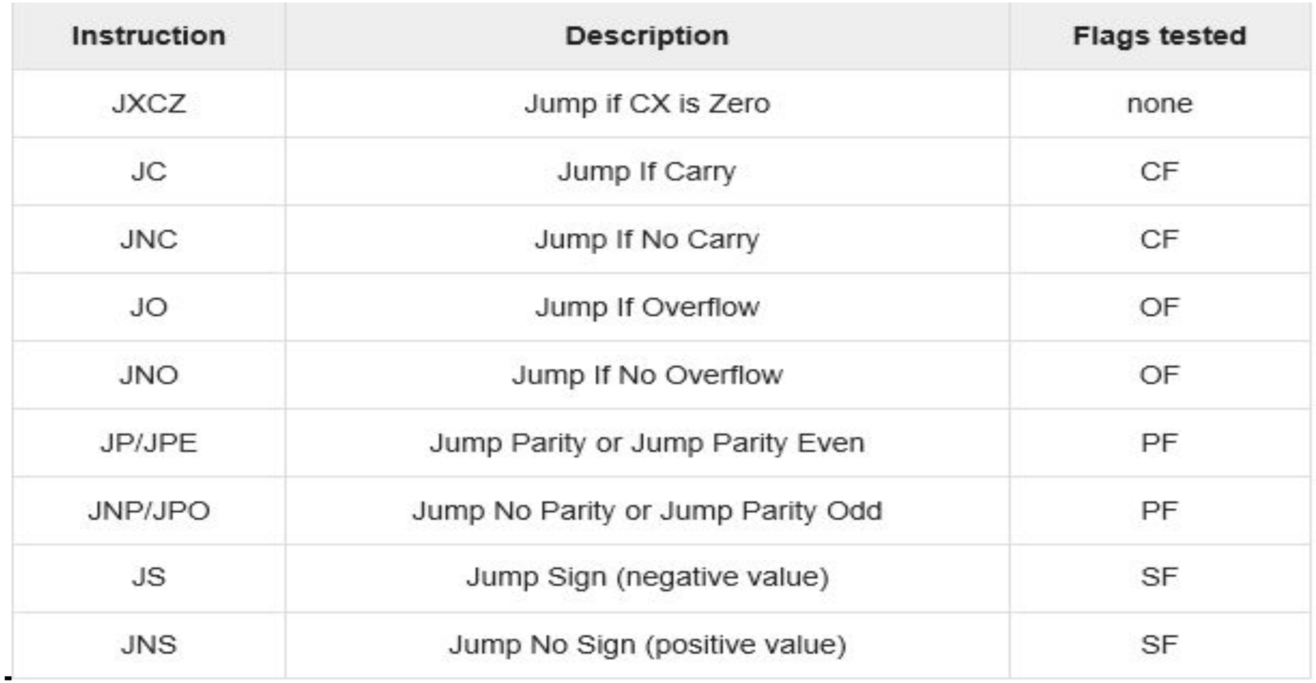
Following are the conditional jump instructions used on unsigned data used for logical operation



Instruction	Description	Flags tested
JE/JZ	Jump Equal or Jump Zero	ZF
JNE/JNZ	Jump not Equal or Jump Not Zero	ZF
JA/JNBE	Jump Above or Jump Not Below/Equal	CF, ZF
JAE/JNB	Jump Above/Equal or Jump Not Below	CF
JB/JNAE	Jump Below or Jump Not Above/Equal	CF
JBE/JNA	Jump Below/Equal or Jump Not Above	AF, CF

List of Conditional Jumps: Flags

The following conditional jump instructions have special uses and check the value of flags

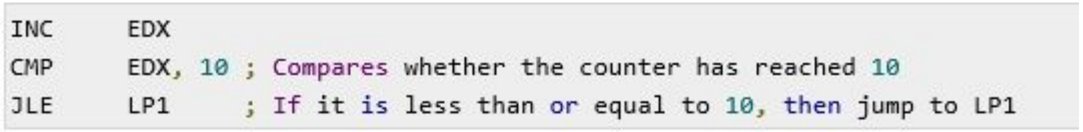


Instruction	Description	Flags tested
JXCZ	Jump if CX is Zero	none
JC	Jump If Carry	CF
JNC	Jump If No Carry	CF
JO	Jump If Overflow	OF
JNO	Jump If No Overflow	OF
JP/JPE	Jump Parity or Jump Parity Even	PF
JNP/JPO	Jump No Parity or Jump Parity Odd	PF
JS	Jump Sign (negative value)	SF
JNS	Jump No Sign (positive value)	SF

CMP Instruction

The CMP instruction compares two operands. It is generally used in conditional execution. This instruction basically subtracts one operand from the other for comparing whether the operands are equal or not.

It does not disturb the destination or source operands. It is used along with the conditional jump instruction for decision making.



```
INC     EDX
CMP     EDX, 10 ; Compares whether the counter has reached 10
JLE     LP1     ; If it is less than or equal to 10, then jump to LP1
```

Loop Instruction

The loop instruction provides some abstraction when handling loops.

It performs the equivalent of these three instructions in one:

- Decrements ecx, `dec ecx`
- Compares ecx with 0, `cmp ecx, 0`
- Jumps back to start of loop label if ecx is not zero, `jnz loop`

Syntax: `loop label`

Error Alert! Jump Out of Range

The target label is referred to as a short-jump. Specifically, that means the target label must be within **128 bytes** from the conditional jump instruction. While this limit is not typically a problem, for very large loops, the assembler may generate an error referring to “jump out-of-range”.

Jump Out of Range: Solution

The unconditional jump (jmp) is not limited in range. If a “jump out-of-range” is generated, it can be eliminated by reversing the logic and using an unconditional jump for the long jump.

Jump Out of Range: Sample Solution

For example, the following code:

```
cmp ecx, 0
jne startOfLoop
```

might generate a “jump out-of-range” assembler error if the label, startOfLoop, is a long distance away.

Jump out of range solution: Cont’d

The error can be eliminated with the following code:

```
cmp ecx, 0
je endOfLoop
jmp startOfLoop
endOfLoop:
```

Which accomplishes the same thing using an unconditional jump for the long jump and adding a conditional jump to a very close label.

`

Crossing the bridge

High Level Meets Low Level

- For loop
- While loop
- Do … while loop

For loops

Used when the number of iteration is fixed.

```
for(int i=1;i<=10;i++){

System.out.println(i);

}
```

For loop in assembly

```
; for
_start
cmp ecx,10   ; i<10
for:
jge _done    ;exit condition.
; do stuff
inc ecx  ; i++
jmp for
_done:
; out of the loop
```

While loop

A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.

While loop starts with the checking of condition. If it evaluated to true, then the loop body statements are executed otherwise first statement following the loop is executed. For this reason it is also called **Entry control loop.**

```
x = 6

while(x>0) {

    ;do something

    x--

}
```

While loop in assembly

```
; while
_start:
ecx,6
test ecx,ecx; same as cmp ecx, 0

jz _done ; jump if zero flag is set
    ; do something
    dec ecx

jmp _start

_done:
; out of the loop
```

Do … while loop

do while loop is similar to while loop with only difference that it checks for condition after executing the statements, and therefore is an example of **Exit Control Loop.**

```
do {
//Statements

x--;

while(x>0)
```

Do … while loop in assembly

```
; do while
_start:
ecx, 6
do_while:
; do stuff
cmp ecx,0
jz _done ; while ecx > 0
dec ecx
jmp do_while
_done:
; out of the loop
```

Questions?