COMPARISON AND CONDITIONAL INSTRUCTIONS

Lecturers: ARD, SWJ Lab Tutors: VDE, WUL

OBJECTIVES

- Able to use comparison and conditional assembly instruction, such as: JZ, JNZ, JC, JNC, JE, JNE, etc
- Able to use CMP in a simple program
- Able to define if – else in assembly language using CMP and JMP instructions
- Able to create a simple program using comparison and conditional instructions

THEORITICAL REVIEW

CMP Instruction

In x86 assembly language we use the CMP instruction to compare integers. Character codes are also integers, so they work with CMP as well. Floating-point values require specialized comparison instructions. The CMP (compare) instruction performs an implied subtraction of a source operand from a destination operand. Neither operand is modified:

```
CMP destination,source
```

CMP uses the same operand combinations as the AND instruction. The CMP instruction changes the Overflow, Sign, Zero, Carry, Auxiliary Carry, and Parity flags according to the value the destination operand would have had if actual subtraction had taken place. When two **unsigned** operands are compared, the Zero and Carry flags indicate the following relations between operands:

CMP Results	ZF	CF
Destination < source	0	1
Destination > source	0	0
Destination = source	1	0

When two **signed** operands are compared, the Sign, Zero, and Overflow flags indicate the following relations between operands:

CMP Results	Flags
Destination < source	SF ≠ OF
Destination > source	SF = OF
Destination = source	ZF = 1

CMP is a valuable tool for creating conditional logic structures. When you follow CMP with a conditional jump instruction, the result is the assembly language equivalent of an IF statement.

Example 1

```
mov ax,5
cmp ax,10       ; ZF = 0 and CF = 1
```

When AX equals 5 and is compared to 10, the Carry flag is set because subtracting 10 from 5 requires a borrow.

Example 2

```
mov ax,1000
mov cx,1000
cmp cx,ax ; ZF = 1 and CF = 0
```

Comparing 1000 to 1000 sets the Zero flag because subtracting the source from the destination produces zero.

Unconditional Jumps

The unconditional jump instruction (jmp) is the asembly version of the goto statement. However there is clearly no shame in using jmp. It is a necessity in assembly language, while goto can be avoided in higher level languages. The basic form of the jmp instruction is:

```
jmp label
```

where label is a label in the program 's text segment . The effect of the jmp statement is that the CPU transfers control to the instruction at the labeled address. This is generally not too exciting except when used with a conditional jump. However, the jmp instruction can jump to an address contained in a register or memory location.

Conditional Jump

A conditional jump instruction branches to a destination label when a status flag condition is true. Otherwise, if the flag condition is false, the instruction immediately following the conditional jump is executed. The syntax is as follows:

```
Jcond destination
```

`cond` refers to a flag condition identifying the state of one or more flags. The following examples are based on the Carry and Zero flags:

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)

CPU status flags are most commonly set by arithmetic, comparison, and boolean instructions. Conditional jump instructions evaluate the flag states, using them to determine whether or not jumps should be taken.

The x86 instruction set has a large number of conditional jump instructions. They are

able to compare signed and unsigned integers and perform actions based on the values of individual CPU flags. The conditional jump instructions can be divided into four groups:

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

Jumps Based on Specific Flag Values

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

Jumps Based on Equality Comparison

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
JRCXZ	Jump if RCX = 0 (64-bit mode)

Jumps Based on Unsigned Comparisons

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)

Jumps Based on Signed Comparisons

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)

TASKS

1. Write instructions that jump to label L1 when the unsigned integer in DX is less than or equal to the integer in CX.
2. Write instructions that jump to label L2 when the signed integer in AX is greater than the integer in CX.
3. Implement the following pseudocode in assembly language.

```
if( val1 > ecx )
     X = 1
else
     X = 2;
```

4. Implement the following pseudocode in assembly language.

```
if( a < b )
     X = 1
else if ( a > c)
     X = 2;
else
     X = 3;
```

5. Implement the following pseudocode in assembly language.

```
if (a > b) AND ( a > c)
     X =  3;
else
     X = 1;
```