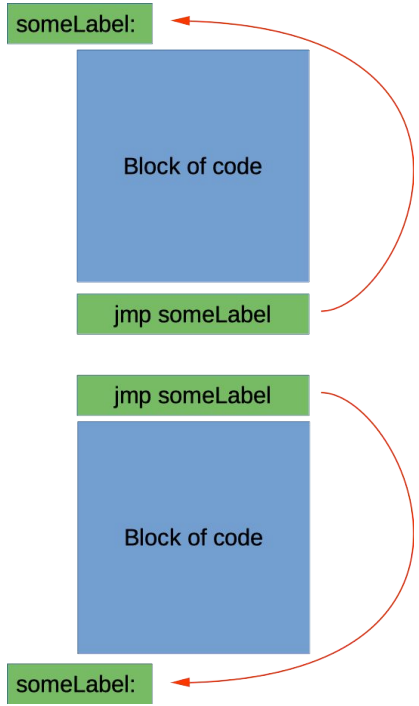
Compares and jumps

CSC 236

Unconditional jump

- Go to some other part of the code
  - .. no matter what.
- Example:
  - jmp label1



Conditional jumps

- Jump based on condition code
  - Either
    - Goto label or
    - “Fall through” to next sequential instruction
- Conditional jumps
  - Typically implemented as a pair of instructions working together
  - The first sets the condition codes
  - The second (jump) is *conditioned* on the results of the other
- Example
  - `cmp  ax,bx`
  - `jle label1   ;if ax<=bx goto label1`

Conditional jumps

- Use the right jumps
- After arithmetic operation:
  you may need to first test for overflow

Conditional jumps — Unsigned

Inst	Name	Arithmetic	Compare	CCs
ja	above	no overflow, ≠0	dest > source	(cf=0) and (zf=0)
jae	above or equal	no overflow	dest ≥ source	cf=0
jb	below	overflow	dest < source	cf=1
jbe	below or equal	overflow or =0	dest ≤ source	(cf=1) or (zf=1)
jc	carry	overflow	dest < source	cf=1
jnc	no carry	no overflow	dest ≥ source	cf=0

Above

- ja (cf = 0) and (zf = 0)
- Unsigned
- CF=0 ⇒ result is correct
- ZF=0 ⇒ result is not zero
- Jump if unsigned result correct and above 0

Below

- jb (cf=1)
- Unsigned
- CF = 1 ⇒ unsigned overflow
  - Addition or subtraction
  - Recall Intel carry rule
- Overflow means
  - dest < source
- Jump if (unsigned) overflow
- MASM synonyms
  - jc ⇔ jb
  - jnc ⇔ jae

8086 carry flag rule

The 8086, on subtraction, always sets the CF to represent the real borrow value

If you use two's complement addition to perform subtraction then the 8086 CF is the inverse of the carry you calculate

(7-9 & 7-10 in Class Notes)

Conditional jumps — Signed

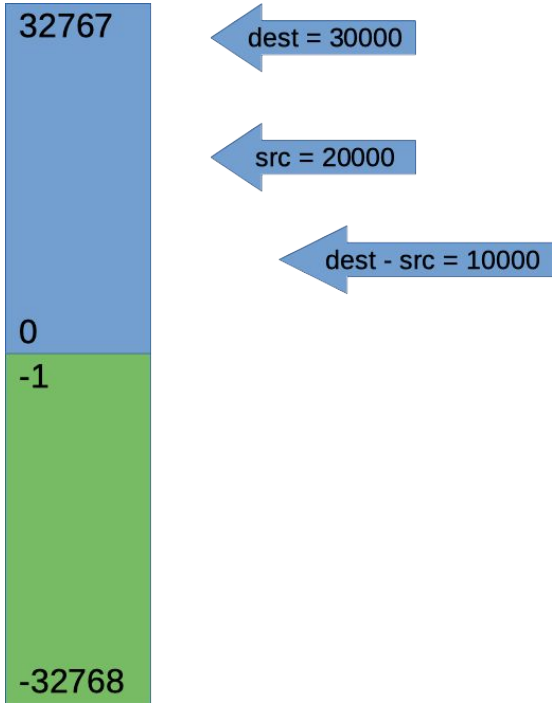
Intr	Name	Arithmetic	Compare	CCs
jg	greater	> 0	dest > source	(sf=of) and (zf=0)
jge	greater or equal	≥ 0	dest ≥ source	sf=of
jl	less	< 0	dest < source	sf≠of
jle	less or equal	≤ 0	dest ≤ source	(sf≠of) or (zf=1)
jo	overflow	overflow		of=1
jno	no overflow	no overflow		of=0
js	sign	leftmost bit = 1		sf=1
jns	no sign	leftmost bit = 0		sf=0

Greater than or equal

- jge (sf=of)
- Signed
- Why does this family of comparisons depend on comparing sf and of?

Comparing dest and src

- What if dest is greater or equal to src?

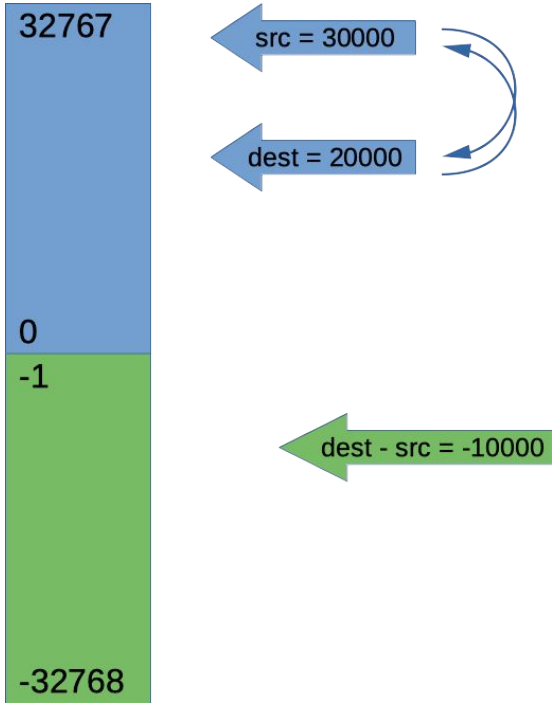


Greater than or equal

- jge (sf=of)
- Signed
- What is meaning of sf=of?
  - If sf = of = 0
  - This is what we'd normally expect if dest ≥ src.
  - The result is non-negative and there was no overflow.

Comparing dest and src

- Or, what if src is larger than dest?

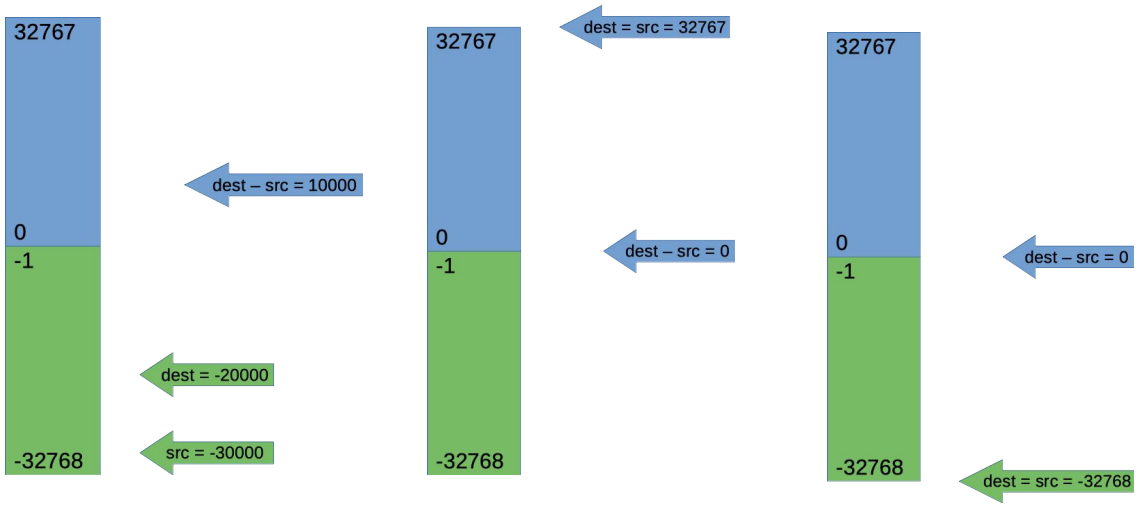


Greater than or equal

- jge (sf=of)
- Signed
- What is meaning of sf=of?
  - If sf = of = 0, then the dest ≥ src.
  - If sf = 1 and of = 0
  - The result is negative (and no overflow)
  - This is what we'd expect if dest < src

Comparing dest and src

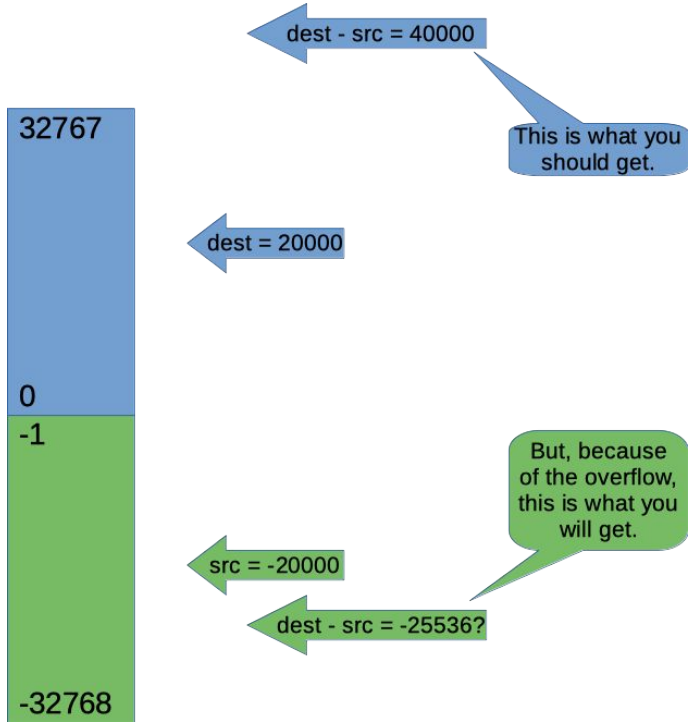
- The sign of the result tells us if dest ≥ src
  - If they're positive
  - If they're negative
  - If they're equal
  - But not in all cases



What if dest is much larger than src?

- If the difference is too large, we'll get overflow ... and the sign of the result will be wrong.

- The same kind of thing can happen if dest is much smaller than src.
- ... we can use this.



Greater than or equal

- jge (sf=of)
- Signed
- What is meaning of sf=of?
  - If sf = 0 and of = 0 ⇒ the result is correct, dest is greater or equal to src.
  - If sf = 1 and of = 0 ⇒ results is correct, dest is less than source
  - If of = 1 ⇒ overflow
    - Sign bit will be wrong
    - If sf = 1 ⇒ result was negative, but only because of overflow
      (so dest is really greater or equal to src)
      (really, just greater than, since you won't get overflow if they're equal)
    - If sf = 0 ⇒ results was non-negative, but only because of overflow
      (so dest is really less than src)

Conditional jumps — Signed

Intr	Name	Arithmetic	Compare	CCs
jg	greater	> 0	dest > source	(sf=of) and (zf=0)
jge	greater or equal	≥ 0	dest ≥ source	sf=of
jl	less	< 0	dest < source	sf≠of
jle	less or equal	≤ 0	dest ≤ source	(sf≠of) or (zf=1)
jo	overflow	overflow		of=1
jno	no overflow	no overflow		of=0
js	sign	leftmost bit = 1		sf=1
jns	no sign	leftmost bit = 0		sf=0

Conditional jumps — Both (signed and unsigned)

Intr	Name	Arithmetic	Compare	CCs
je	equal	= 0	dest = source	zf=1
jne	not equal	≠ 0	dest ≠ source	zf=0

Unsigned Conditional Jumps (these use the terms *above* and *below*)

Instruction	Name	Condition tested: Arithmetic	Condition tested: Compare	Jump taken if flags have these values
ja	above	no overflow and result ≠ 0	dest > source	(cf = 0) and (zf = 0)
jae	above or equal	no overflow	dest ≥ source	(cf = 0)
jb	below	overflow	dest < source	(cf = 1)
jbe	below or equal	overflow or result = 0	dest ≤ source	(cf = 1) or (zf = 1)
jc	carry	overflow	dest < source	(cf = 1)
jnc	no carry	no overflow	dest ≥ source	(cf = 0)

Signed Conditional Jumps (these use the terms *greater* and *less*)

Instruction	Name	Condition tested: Arithmetic	Condition tested: Compare	Jump taken if flags have these values
jg	greater	result > 0	dest > source	(sf = of) and (zf = 0)
jge	greater or equal	result ≥ 0	dest ≥ source	(sf = of)
jl	less	result < 0	dest < source	(sf ≠ of)
jle	less or equal	result ≤ 0	dest ≤ source	(sf ≠ of) or (zf = 1)
jo	overflow	overflow		(of = 1)
jno	no overflow	no overflow		(of = 0)
js	sign	left most bit = 1		(sf = 1)
jns	no sign	left most bit = 0		(sf = 0)

Signed and Unsigned Conditional Jumps

Instruction	Name	Condition tested: Arithmetic	Condition tested: Compare	Jump taken if flags have these values
je	equal	result = 0	dest = source	(zf = 1)
jne	not equal	result ≠ 0	dest ≠ source	(zf = 0)

Class notes
Page 6-16

Example — unsigned

```
a    db  ??   ;unsigned byte
b    db  ??   ;unsigned byte
c    db  ??   ;unsigned byte
```

```
                    mov  al,[a]   ;load a
                    add  al,[b]   ;calc a+b
                    ???  err      ;handle overflow
                    mov  [c],al   ;else c=a+b

                    …

               err: …       ;code to handle error
```

Example — unsigned

```
a    db  ??   ;unsigned byte
b    db  ??   ;unsigned byte
c    db  ??   ;unsigned byte
```

```
                    mov  al,[a]   ;load a
                    add  al,[b]   ;calc a+b
                    jc   err      ;handle overflow
                    mov  [c],al   ;else c=a+b

                    …

              err:  …       ;code to handle error
```

Example — signed

```
a    db  ??   ;signed byte
b    db  ??   ;signed byte
c    db  ??   ;signed byte
```

```
                    mov  al,[a]   ;load a
                    add  al,[b]   ;calc a+b
                    ???  err      ;handle overflow
                    mov  [c],al   ;else c=a+b

                    …

              err:  …       ;code to handle error
```

Example — signed

```
a    db  ??   ;signed byte
b    db  ??   ;signed byte
c    db  ??   ;signed byte
```

```
                    mov  al,[a]   ;load a
                    add  al,[b]   ;calc a+b
                    jo   err      ;handle overflow
                    mov  [c],al   ;else c=a+b

                    …

              err:  …       ;code to handle error
```

Grade calculation

If grade is zero then goto trouble

```
grade     db ?          ;unsigned byte 0..100

cmp       [grade],0     ;grade ? 0
je        trouble       ;if grade is 0 goto trouble
```

Speeding

If speed is greater than 55 then goto ticket

```
speed   db ?          ;unsigned byte (max 255MPH)

cmp     [speed],55    ;speed ? 0
ja      ticket        ;jump if speed>55
```

Water

If temp is 212 or more then goto boil
else if temp is 32 or less then goto freeze
else goto nice

```
temp    dw    ?         ;signed word -50 ... +300
```

Water

If temp is 212 or more then goto boil
else if temp is 32 or less then goto freeze
else goto nice

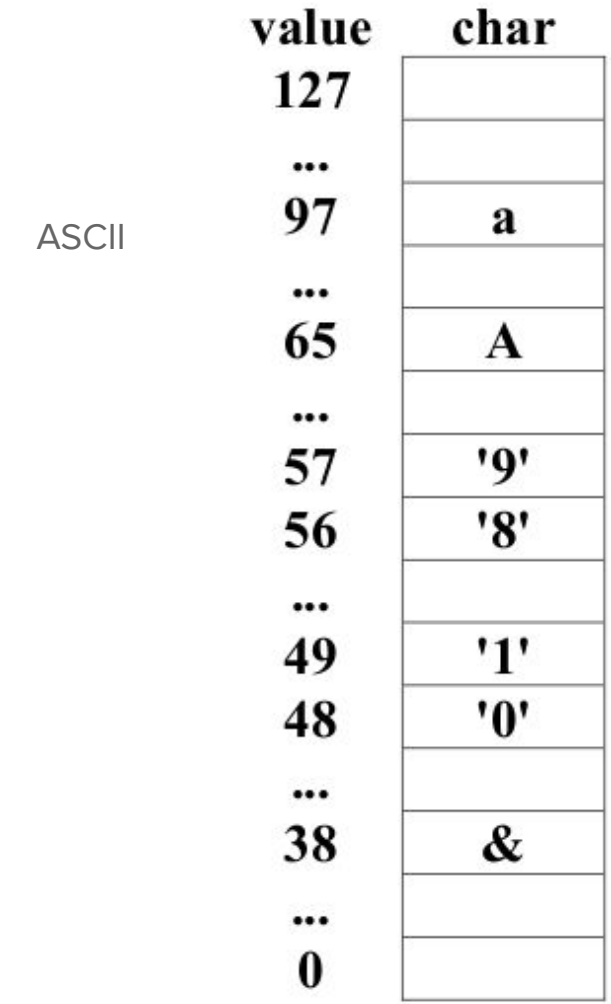
```
temp    dw    ?        ;signed word -50 ... +300

cmp     [temp],212     ;temp ? 212
jge     boil           ;boiling >= 212
cmp     [temp],32      ;temp ? 32
jle     freeze         ;freezing <= 32
jmp     nice           ;freeze <temp< boil
```

Range checking

Is byte variable char an ASCII digit?

```
cmp   [char],'0'  ;test lower bound
jb    not         ;below - not digit
cmp   [char],'9'  ;test upper bound
ja    not         ;above - not digit
jmp   yes
```

ASCII



Setting condition codes

Suppose:

- `add`
- `cbw`
- `mov`
- `push`
- `pop`
- `jxx`

Which instruction sets the condition code tested by jxx

- Notes specify which CCs an instruction sets
- pop does not
- push does not
- mov does not
- cbw does not
- add does
- So jxx checks CC set by add
- Is this a good idea?
- Comment would be very useful

Setting condition codes — multiple jumps

```
add     ax,bx          ; ax = ax + bx  (signed)
jo      error          ; test signed overflow
je      is_zero        ; test result = 0
jg      is_greater     ; test result > 0
jl      is_less_than   ; test result < 0
```

Banking application

```
balance  amount in checking account
deposit  amount to be deposited
check    amount to be withdrawn
```

Banking application

```
balance  dw  ?  ;signed
deposit  dw  ?  ;??
check    dw  ?  ;??
```

- Allow balance to go negative
  - Overdraft
- If balance is signed what about
  - Deposit
  - Check

- unsigned ? unsigned ⇒ jc
- signed ? signed ⇒ jo
- unsigned ? signed ⇒ ???

Banking application

```
balance  dw  ?  ;signed
deposit  dw  ?  ;??
check    dw
```

- Allow balance to go negative

C program

```
unsigned U = 10;
if (U < -1) printf ("U less than -1 ");
```

Output:

U less than -1

- at about
- → jc
- signed ? signed ⇒ jo
- unsigned ? signed ⇒ ???

Banking application

```
balance  dw  ?  ;signed
deposit  dw  ?  ;signed
check    dw  ?  ;signed
```

- Make deposit
  - balance += deposit
  - if overflow then goto error1

Banking application

```
balance  dw  ?  ;signed
deposit  dw  ?  ;signed
check    dw  ?  ;signed
```

- Make deposit
  - balance += deposit
  - if overflow then goto error1

```
mov  ax,[deposit]
add  [balance],ax
jo   error1
```

Banking application

```
balance  dw  ?  ;signed
deposit  dw  ?  ;signed
check    dw  ?  ;signed
```

- Cash check
  - if balance < 0 then goto bounce
  - balance -= check
  - if overflow then goto error2
  - if balance < 0 then goto interest

Banking application

```
balance  dw  ?  ;signed
deposit  dw  ?  ;signed
check    dw  ?  ;signed
```

- Cash check
  - if balance < 0 then goto bounce
  - balance -= check
  - if overflow then goto error2
  - if balance < 0 then goto interest

```
cmp     [balance],0
jl      bounce
mov     ax,[check]
sub     [balance],ax
jo      error2
jl      interest
```

Testing

```
cmp     [balance],0
jl      bounce
mov     ax,[check]
sub     [balance],ax
jo      error2
jl      interest
```

- Write tests for all cases
- But
  - Unable to test error2
  - Why not?

```
balance -= check
```

- Smallest value for balance is 0
- Largest value for check is 32767
- -32767 does not overflow unsigned word

Class notes have

- Assembly language templates for many HLL constructs
  - if … then … else
  - for
  - while
  - switch
  - Pages 7-13 — 7-16
- Sample coding exercises
  - Pages 7-11

C code example

Who is responsible for knowing whether data is signed or unsigned?

```
int  a;
if (a < 100)  a++;
```

```
    cmp  [a],100
    jge  $I1
    inc  [a]
$I1:
```

```
unsigned  b;
if (b < 100)  b++;
```

```
    cmp   [b],100
    jae     $I1
    inc   [b]
$I1:
```