Basic Assembly Instructions

Ned Nedialkov

McMaster University
Canada

SE 3F03
January 2014

Outline

Multiplication

Division

FLAGS register

Branch Instructions

If statements

Loop instructions

Multiplication

- **mul** is for unsigned integers
- **imul** is for signed integers
- $255 \times 255 = 65025$ if unsigned
  255 is 1 if signed
- `FFh = 1111|1111`
  - as unsigned is 255
  - as signed is `1|1111111 = -1`
- Two's complement representation
  - first bit 1 means $-$; 0 means $+$
  - flip all the bits
  - add `1`

mul

- **mul** `source`
  - `source` can be register or memory
  - the other operand is implicit

source	other operand	result
byte	**AL**	**AX**
word	**AX**	**DX**:**AX**
dword	**EAX**	**EDX**:**EAX**

imul

- **imul** `source`
  - `source` can be register or memory
  - the other operand is implicit
- **imul** `dest, source`
- **imul** `dest, source1, source2`

See Table 2.2 for details

Division

- **div** is for unsigned integers
- **idiv** is for signed integers
- both work the same way
- **div** `source`
  - `source` can be register or memory

source	division	quotient	remainder
byte	**AX**/`source`	**AL**	**AH**
word	(**DX**:**AX**)/`source`	**AX**	**DX**
dword	(**EDX**:**EAX**)/`source`	**EAX**	**EDX**

Do not forget to initialize **DX** or **EDX**

FLAGS register

- Contains various flags
- **cmp** `a, b`
  - subtracts $a - b$
  - does not store the result
  - sets flags
- For unsigned integers
  - `ZF` zero flag
  - `CF` carry flag
- For signed integers
  - `ZF` zero flag
  - `OF` overflow flag; 1 when the result overflows
  - `SF` sign flag; 1 when the result is negative

cmp

- Unsigned integers

cmp a,b

a−b	ZF	CF
=0	1	0
>0	0	0
<0	0	1

- Signed integers

cmp a,b

a−b	ZF	SF,OF
=0	1	
>0	0	SF=O, SF=OF
<0	0	SF=1, SF!=OF

Branch Instructions

Unconditional branches

- **jmp** `label`
- **call** `label`
  - unconditional branch
  - like `goto label`

Conditional branches

- `jxx label`
- check flags
- if true, branch (transfer execution control) to `label`
- otherwise, continue from the next statement

- `jxx` **short** `label`
  - the jump is $\pm$128 bytes from the current location
  - advantage: the offset is 1 byte
- `jxx` **near** `label`
  - the jump is to any location within a segment
  - `label` is 32 bit
  - default, same as `jxx label`
- `jxx` **word** `label`
  - 16-bit label
- `jxx` **far** `label`
  - outside a segment

Do **cmp** `a,b`
Then

if	signed	unsigned
`a=b`	**je**	**je**
`a!=b`	**jne**	**jne**
`a<b`	**jl** , **jnge**	**jb**, **jnae**
`a>b`	**jg**, **jnle**	**ja**, **jnbe**
`a>=b`	**jge**, **jnl**	**jae**, **jnb**

For more instructions, see the text

If statements

```
if (condition) {
/* then block */
}
else {
/* else block */
}
```

Can be translated as

```
        ;; code that sets flags
        ;; e.g. cmp a,b
        jxx else_block
        ;; code in then block
        jmp end_if
else_block:
        ;; code in else block
endi_if:
```

`jxx` is a suitable branch instruction

```
if (condition) {
/* then block */
}
```

Can be translated as

```
        ;; code that sets flags
        ;; e.g. cmp a,b
        jxx end_if
        ;; code in then block
endi_if:
```

`jxx` is a suitable branch instruction

Examples

```
sum=0;
i=i-1;
if (i>0) sum++;
```

Can be translated into

```
        ;; assume i is in ecx
        mov eax, 0                  ;sum=0
        dec ecx                     ;i=i-1
        jz end_if
        inc eax                     ;sum++
end_if:
```

```
if (eax>=5)
    ebx=1
else
    ebx=2
```

Can be translated into

```
        cmp eax, 5
        jge then_block
        mov ebx, 2
        jmp next
then_block:
        mov ebx, 1
next:
```

...or into

```
        cmp eax, 5
        jnz else_block
        mov ebx, 1
        jmp next
else_block:
        mov ebx, 2
next:
```

Loop instructions

loop instruction

Example

```
sum = 0;
for (i=10; i>0; i--)
      sum += i;
```

Can be translated into

```
        mov eax, 0                  ;sum=0
        mov ecx, 10                 ;ecx=10, loop counter
loop_start:
        add eax, ecx                ;sum+=i
        loop loop_start             ;ecx--, goto loop_start
```

Example

```
sum = 0;
for (i=1; i<=10; i++)
      sum += i;
```

Is the following a correct translation?

```
        mov ebx, 1
        mov eax, 0                  ;sum=0
        mov ecx, 10                 ;ecx=10, loop counter
loop_start:
        add eax, ebx                ;sum+=i
        inc ebx
        loop loop_start             ;ecx--, goto loop_start
```

loop

- **loop** `start_loop` same as
  - decrement **ecx** by 1
  - if **ecx**`!=0` goto `start_loop`
- **loope** `start_loop`
- **loopz** `start_loop` same as
  - decrement **ecx** by 1
  - if **ecx**`!=0` and `ZF==1` goto `start_loop`
- **loopne** `start_loop`
- **loopnz** `start_loop` same as
  - decrement **ecx** by 1
  - if **ecx**`!=0` and `ZF==0` goto `start_loop`

`ZF` unchanged if **ecx**`=0`

While loops

Example

```
while (condition) {
  /* body of the while loop */
 }
```

Can be translated into

```
while:
        ;; code that sets flags
        jxx end_while              ;branch if false
        ;; code in the while body
        jmp while
end_while:
```

Do-while loops

Example

```
do  {
  /* body of the loop */
 } while (condition)
```

Can be translated into

```
do:
        ;; body of the loop
        ;; code that sets flags
        jxx do                      ;branch if true
end_while:
```