Multiplication and Division Instructions

Module 9
CS 272
Sam Houston State University
Dr. Tim McGuire

Multiplication instructions

- **`imul`** ***source***
  - Integer (signed) multiply
- **`mul`** ***source***
  - Unsigned multiply

Byte and Word Multiplication

- If two bytes are multiplied, the result is a 16-bit word
- If two words are multiplied, the result is a 32-bit ***doubleword***
- For the byte form, one number is contained in the source and the other is assumed to be in **`al`** -- the product will be in **`ax`**
- For the word form, one number is contained in the source and the other is assumed to be in **`ax`** -- the most significant 16 bits of the product will be in **`dx`** and the least significant 16 bits will be in **`ax`**

Multiply Instruction

Signed Multiply

`IMUL` ***`source`***

Unsigned Multiply

`MUL` ***`source`***

- *source* can be a register or memory location (not a constant)

- Byte form
  - AX=AL***source*
- Word form
  - DX:AX=AX***source*
- CF=OF
  - 1 if answer size > op size
  - 0 otherwise
- SF, ZF, AF, and PF
  - Undefined

Examples

- If **ax** contains **0002h** and **bx** contains **01FFh**

```
mul    bx
  dx = 0000h        ax = 03FEh
```

- If **ax** contains **0001h** and **bx** contains **FFFFh**

```
mul    bx
  dx = 0000h        ax = FFFFh
imul   bx
  dx = FFFFh        ax = FFFFh
```


Another IMUL Example

```
mov word [AWORD],-136
mov AX,6784
imul word [AWORD]
```

- AWORD contains 0FF78h, AX is 1A80h
- After IMUL
  - DX:AX contains 0FFF1EC00h (-922624d)
  - CF=OF=1 (result requires a doubleword)

Another MUL Example

```
mov word [AWORD],-136
mov AX,6784
mul word [AWORD]
```

- AWORD is 0FF78h (65400d), AX is 1A80h
- After MUL
  - DX:AX contains 1A71EC00h (443673600d)
  - CF=OF=1 (result requires a doubleword)

One More IMUL Example

```
mov AX,-1
mov DX,2
imul DX
```

- AX is 0FFFFh, BX is 0002h
- After IMUL
  - DX:AX contains 0FFFFFFFEh (-2d)
  - CF=OF=0 (result still fits in a word)

Application: Formatted Decimal Input

- Input a number (sequence of decimal digits)
  - Implement as a procedure
  - Skip leading whitespace, stop processing at next whitespace (tab, space, or newline)
  - Return result in AX
  - Return with CF=1 if error occurred
    - Errors: overflow, illegal digit, no digit
  - Assume unsigned data for this example

Conversion Algorithm

- skip whitespace
- v = 0
- while (digit d is not whitespace)
  - if (illegal) set illegal flag
  - otherwise process digit
    - v = v*10 + d
  - if (overflow) set overflow flag

```
nextDigit:
…
 mov ax,10
 mul [valueSoFar]
 jo  setOF
 add ax,[digitValue]
 jnc nextDigit
setOF:
 inc [OFFlag]
 jmp nextDigit
```

Division instructions

- **`cbw`**
  - convert byte to word
- **`cwd`**
  - convert word to doubleword
- **`div`** ***source***
  - unsigned divide
- **`idiv`** ***source***
  - integer (signed) divide

Byte and Word Division

- When division is performed, there are two results, the quotient and the remainder
- These instructions divide 8 (or 16) bits into 16 (or 32) bits
- Quotient and remainder are same size as the divisor
- For the byte form, the 8 bit divisor is contained in the source and the dividend is assumed to be in **`ax`** -- the quotient will be in **`al`** and the remainder in **`ah`**
- For the word form, the 16 bit divisor is contained in the source and the dividend is assumed to be in **`dx:ax`** -- the quotient will be in **`ax`** and the remainder in **`dx`**

Examples

- If **dx** = **0000h**, **ax** = **00005h**, and **bx** = **0002h**

```
div    bx
   ax = 0002h        dx = 0001h
```

- If **dx** = **0000h**, **ax** = **0005h**, and **bx** = **FFFEh**

```
div    bx
   ax = 0000h        dx = 0005h
idiv   bx
   ax = FFFEh        dx = 0001h
```


Divide Overflow

- It is possible that the quotient will be too big to fit in the specified destination (**al** or **ax**)
- This can happen if the divisor is much smaller than the dividend
- When this happens, the program terminates and the system displays the message "**Divide Overflow**"

Sign Extension of the Dividend

- Word division
  - The dividend is in **dx:ax** even if the actual dividend will fit in **ax**
  - For **div**, **dx** should be cleared
  - For **idiv**, **dx** should be made the sign extension of **ax** using **cwd**
- Byte division
  - The dividend is in **ax** even if the actual dividend will fit in **al**
  - For **div**, **ah** should be cleared
  - For **idiv**, **ah** should be made the sign extension of **al** using **cbw**

Divide Instruction

Signed Divide

IDIV *divisor*

Unsigned Divide

DIV *divisor*

- *divisor* can be a register or memory location (not a constant)

- Byte form
  - AL=AX / *divisor*
  - AH=AX % *divisor*
- Word form
  - AX=DX:AX / *divisor*
  - DX=DX:AX % *divisor*
- All Flags undefined
- For IDIV
  - sign of dividend = sign of remainder

Formatted Numeric Output

- Want to produce a sequence of characters representing a number in some representation scheme
  - decimal, hex, octal, binary are common representation schemes
  - decimal format usually implies a sign might be shown if the number is negative (assuming we are displaying a signed value)

Generating the Digits

- `least_significant_digit = value % base`
- we can reuse this computation for the next digit if we modify value:

  ```
  value = value / base
  ```

- The process repeats until a 0 is obtained or the desired number of digits determined
- Problem: we generate the digits in the wrong order for display

Reversing the Digits

Solution A

- Push digits on stack as they are discovered
- Pop each digit off the stack and display it

Solution B

- Store digits in an array as they are discovered
- Display digits from the array in the opposite order

A Decimal Output Procedure

- Given an unsigned number in BX
- Display the decimal representation on the screen

DEC_OUT Procedure

- See hexdec.asm