CS 271 Lecture #5b

Writing a MASM program

Rules & regulations

Syntax and semantics

MASM Instructions

- For now, know how to use
  - mov
  - add
  - sub
  - mul
  - div
  - call
- Some instructions use implied operands
- See textbook (Appendix) or on-line *Instructions*

Easy Instructions

```
mov     op1, op2    ;op2 is copied to op1
add     op1, op2    ;op2 is added to op1
sub     op1, op2    ;op2 is subtracted from op1
inc     op1         ; add 1 to op1
dec     op1         ; subtract 1 from op1
```

- For 2-operand instructions the first operand is the destination and the second operand is the source
- 2-operand instructions require at least one of the operands to be a register (or *op2* must be literal).
  - Note: *op1* can not be a literal !!
  - Discussion question #5:
    - Why can't *op1* be a literal ?

Instructions with implied operands

mul implied operand must be in EAX

```
mul  op2          ; result is in EDX:EAX
```

Example:

```
mov   eax,10
mov   ebx,12
mul   ebx       ; result is in eax (120),
                ;   with possible
                ;   overflow in edx
                ; edx is changed!
```

Instructions with implied operands

div implied operand is in EDX:EAX

so **extend EAX into EDX** before division

```
div   op2          ; quotient is in EAX
                   ; remainder is in EDX
```

Example:

```
mov  eax,100
cdq                ;extend the sign into edx
mov  ebx,9
div  ebx           ; quotient is in eax (11)
                   ; remainder is in edx (1)
```

Operand notation (see instruction list)

Operand	Description
r8	8-bit general-purpose register: AL, AH, BL, BH, CL, CH, DL, DH
r16	16-bit general-purpose or multi-purpose register: AX, BX, CX, DX, SI, DI, BP, SP
r32	32-bit general-purpose or multi-purpose register: EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP
reg	any general-purpose or multi-purpose register
accum	AL, AX, or EAX (depending on operand size)
mem	8-bit, 16-bit, or 32-bit memory location (depending on operand size)
segreg	16-bit segment register: SS, CS, DS, ES, FS, GS
r/m8	8-bit register or memory location
r/m16	16-bit register or memory location
r/m32	32-bit register or memory location
imm8	8-bit literal value
imm16	16-bit literal value
imm32	32-bit literal value
imm	8-bit, 16-bit, or 32-bit literal value (depending on operand size)

Examples

Syntax	Examples
MOV *mem,accum*	`mov total,eax` `mov response,al`
MOV *accum,mem*	`mov al,char` `mov eax,size`

Syntax	Examples
MOV *mem,imm*	`mov color,7` `mov response,'y'`

Syntax	Examples
MOV *reg,imm*	`mov ecx,256` `mov edx,OFFSET myString`

Notes:

accum means "eax or some valid part of eax"

imm means "a literal or constant"

Examples

Syntax	Examples

MOV *reg,reg*	`mov dh,bh`
	`mov edx,ecx`
	`mov ebp,esp`

MOV *mem,reg*	`mov count,ecx` `mov num1,bx`
MOV *reg,mem*	`mov ebx,pointer` `mov al,response`

Syntax	Examples
MOV *sreg,reg16*	`mov ds,ax`
MOV *sreg,mem16*	`mov es,pos1`
MOV *reg16,sreg*	`mov ax,ds`
MOV *mem16,sreg*	`mov stack_save,ss`

Notes:

mem8 means “BYTE”

mem16 means “WORD”

mem32 means “DWORD”

sreg means CS, DS, ES, FS, GS, or SS

Invalid MOV statements

```
.data
bVal   BYTE    100
bVal2  BYTE    ?
wVal   WORD    2
dVal   DWORD   5

.code
   mov ds,45          immediate move to DS not permitted
   mov esi,wVal       size mismatch
   mov eip,dVal       EIP cannot be the destination
   mov 25,bVal        immediate value cannot be destination
   mov bVal2,bVal     memory-to-memory move not permitted
```

Libraries

- We will use Irvine's library (for now) to handle the really awful stuff.
  - input / output
  - screen control
  - timing
  - etc.
- Check IrvineLibHelp, or find the descriptions in your textbook.

Library Procedures - Overview p1

- Clrscr - Clear the screen
  - Preconditions: none
  - Postconditions: screen cleared and cursor is at upper left corner

- Crlf - New line
  - Preconditions: none
  - Postconditions: cursor is at beginning of next new line

Library Procedures - Overview p2

- ReadInt - Reads an integer from keyboard, terminated by the *Enter* key.
  - Preconditions: none
  - Postconditions: value entered is in EAX

- ReadString - Reads a string from keyboard, terminated by the *Enter* key.
  - Preconditions: OFFSET of memory destination in EDX
    Size of memory destination in ECX
  - Postconditions: String entered is in memory
    Length of string entered is in EAX

Library Procedures - Overview p3

- WriteInt, WriteDec - Writes an integer to the screen.
  - Preconditions: value in EAX
  - Postconditions: value displayed
  - WriteInt displays +/-
- WriteString - Writes a null-terminated string to the screen.
  - Preconditions: OFFSET of memory location in EDX
  - Postconditions: string displayed

Calling a Library Procedure

- The INCLUDE directive copies the procedure prototypes (declarations) into the program source code.

- Call a library procedure using the CALL instruction.

In-line Comments

- Start with ;
- May be on separate line or at end of a line
- Use comments to clarify lines or sections
- Preferred …

```
; Calculate the number of students on-line today.
            mov     eax,size
            sub     eax,absent
            mov     present,eax
```

- OK …

```
            mov     eax,size        ;start with class size
            sub     eax,absent      ;subtract absentees
            mov     present,eax     ;number present
```

- Terrible …

```
            mov     eax,size        ;move size into eax
            sub     eax,absent      ;subtract absent from eax
            mov     present,eax     ;move eax to present
```

Programming Project #1

- Problem definition on the course website
- Simple problem
  - Getting accustomed to MASM syntax and semantics
- Submit via your ENGR account
  - http://engr.oregonstate.edu/teach
- Due before midnight, second Sunday