x86 Assembly Language

too much fun for just one day

prepared by

jonathan lung

http://www.cs.toronto.edu/~lungj

Winter 2006

Scope of Discussion

- 16-bit x86 programming
- A little bit of context
- The low down
- A short example
- Questions & Answers

Assembly Language

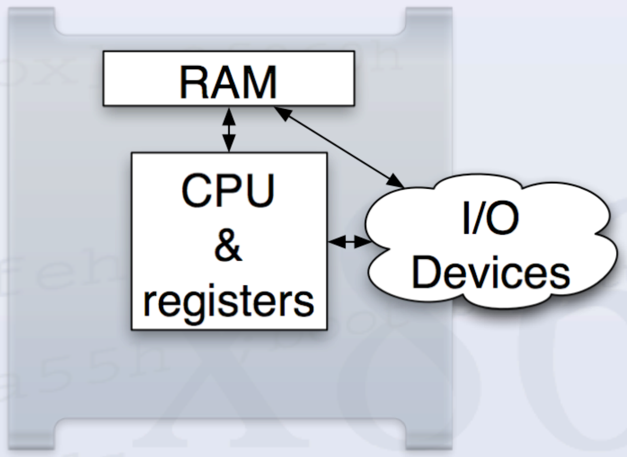
- Early programming language
- Low level
- Assembled by assemblers such as
  - Flat assembler (FASM)
  - Microsoft Macro Assembler (MASM)
  - Netwide Assembler (NASM)
  - Borland Turbo Assembler (TASM)
- In-line assembly language support

The Scoop

- This lecture is not about
  - Computer hardware
  - Cracking
  - Writing mal-ware
  - The merits of assembly language
  - Writing optimized assembly code
- This lecture is about
  - Understanding system tools
  - Demystifying language functions

The Fundamental Fact

- A program is nothing more than a sequence of instructions telling a computer how to move bits around



Opcodes

- One-to-one correspondence
- Written as *mnemonics*
- Take the form

```
MNEMONIC target, source
```

E.g. `ADD AX, BX`

Targets and Sources

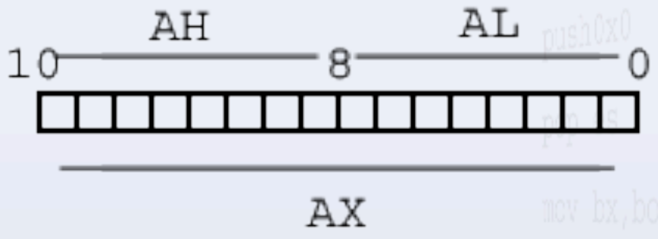
- Immediate
- Register
- Memory
- Stack

Immediate

- Constant value
- Can act as source

Registers

- Four general purpose registers
  - AX
  - BX
  - CX
  - DX
- 16 bits long
- Sub-dividable into halves



Registers

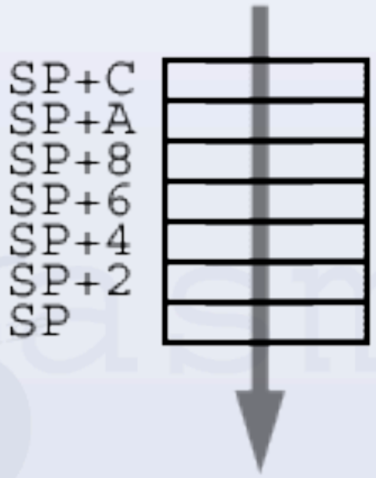
- Four segment registers
  - CS
  - DS
  - ES
  - SS

Memory

- Memory address written as
  `SEGMENT:OFFSET`
- Dereference offset with square brackets
  `CS:[C494]`
- DS is implicit when not specified
  `[1337]` is the same as `DS:[1337]`

Stack

- First in, last out (FILO)
- Top of the stack is at `SS:SP`
- Grows downwards
- No bounds checking



Operations

- Arithmetic
- Logic
- Bit manipulation
- Comparisons and jumps
- Function calls
- Other

Arithmetic

- ADD
- SUB
- MUL
- DIV

Arithmetic

- ADD
- SUB
- MUL
- DIV

```
→  ADD AX, 5      AX = 0003
   …
```

0E

Arithmetic

- ADD
- SUB
- MUL
- DIV

```
      ADD AX, 5      AX = 0008
  →   …
```

0E

Logic

- AND
- OR
- XOR
- NOT

Logic

- AND
- OR
- XOR
- NOT

```
→ AND CH, DL      CH = 11111111  DL = 00000010
  NOT DL
  …
```

0F

operations

Logic

arithmetic **logic** bit manipulations comparisons and jumps function calls other

- AND
- OR
- XOR
- NOT

```
    AND CH, DL      CH = 00000010   DL = 00000010
──► NOT DL
    …
```

0F

operations

Logic

arithmetic | **logic** | bit manipulations | comparisons and jumps | function calls | other

- AND
- OR
- XOR
- NOT

```
AND CH, DL      CH = 00000010  DL = 11111101
NOT DL
…
```

$\longrightarrow$

0F

Bit Manipulation

- SHL/SHR
  - E.G. SHL AL, 1

```
101101010
 ↙↙↙↙↙↙↙↙
01101010  ;(SHL by 1)
```

Comparisons and Jumps

- JMP
- CMP
- Jxx

Function Calls

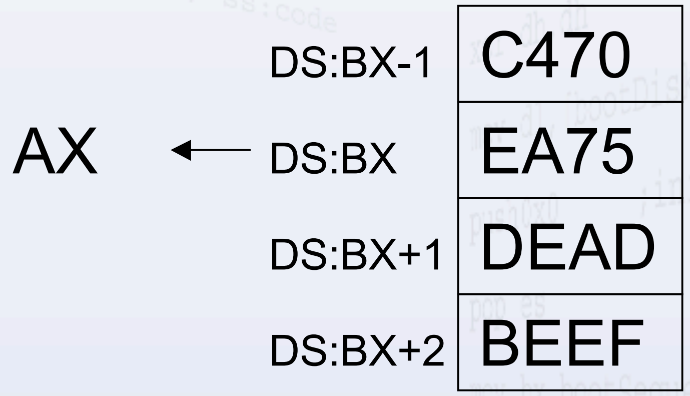
- CALL
- RET

Other

- MOV
  - E.g. MOV AX, BX $AX \leftarrow BX$

Other

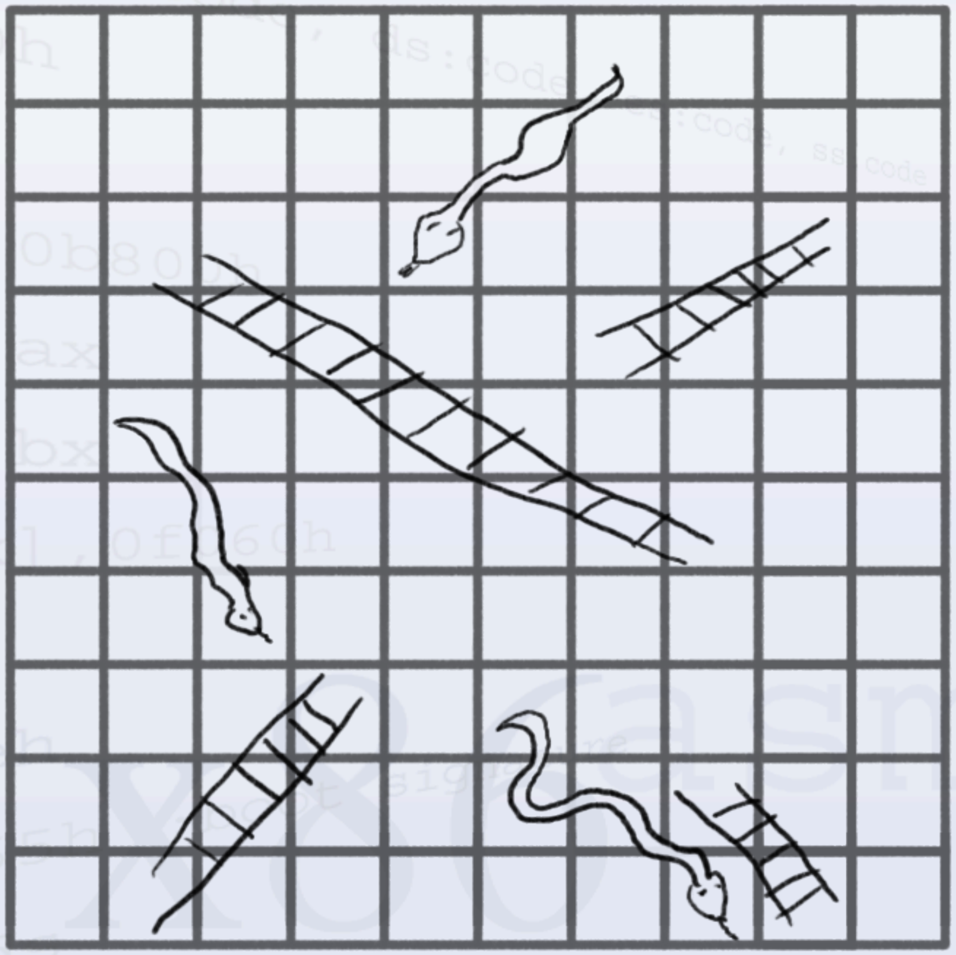
- MOV
  - E.g.
    ```
    MOV   AX, BX
    MOV   AX, [BX]
    ```



Other

- MOV
  - E.g.
    ```
    MOV   AX, BX
    MOV   AX, [BX]
    ```
- PUSH/POP
  - E.g.
    ```
    PUSH BX
    POP AX
    ```
- IN/OUT
- NOP

Snakes And Ladders



Snakes And Ladders

```
            MOV   BX, 0  ;current location
            MOV   CX, 0  ;# moves so far
NEXT_FLIP:  CALL  GETNEXTCOINFLIP
            ADD   BX, AX ;# spaces to move
            ADD   CX, 1
            ADD   BX, DS:[BX]
            CMP   BX, 64 ;64h=100 base 10
            JL    NEXT_FLIP
HANG:       JMP HANG
```

Questions & Answers

- For more information…
  - IA-32 Intel Architecture Software Developer's Manual
  - The Peter Norton Programmer’s Guide to the IBM PC
  - Inside the IBM PC