Nội dung chính

- Thanh ghi trạng thái processor
- Các lệnh sao chép dữ liệu
- Các lệnh số học và logic
- Các lệnh thao tác với bit
- Các lệnh điều khiển
- **Các câu lệnh với chuỗi**
- Các câu lệnh khác

Câu lệnh

- X86 cung cấp một số lệnh liên quan chuỗi byte, word, dword
  - Lệnh: Movs_, Stos_, Lods_, Cmps_, Scas_,…
- Chỉ ra kích thước dữ liệu bởi hậu tố [b, w, d]
  - Kiểu dword có từ 386
- Thường dùng kèm các chỉ thị lặp REP, REPZ..
  - CX chứa số lần lặp, sẽ giảm tự động đi 1 đơn vị
- Thanh ghi SI, DI thay đổi phụ thuộc cờ hướng
  - Thay đổi cờ hướng: CLD, STD

Câu lệnh **MOVSB**, **MOVSW**, **MOVSD**

[REP] **MOVSB/ MOVSW/ MOVSD**

- Copy dữ liệu từ vị trí bộ nhớ được xác định bởi **DS:SI** sang vị trí trong **ES:DI**
  - Thực hiện: ES:[DI] ←DS:[SI]
  - **SI**, **DI** được tăng /giảm tự động 1/2/4 byte
    - Tăng khi DF=0=UP; Giảm khi DF=1=DOWN
- Đặt chỉ thị **REP** trước câu lệnh
  - CX xác định **số lần** lặp. Kết thúc lặp CX = 0

Ví dụ: Xóa phần tử đầu tiên của vector

```
.model tiny    ;
.data
   Vec DB "HHello world$"
   len = $-Vec  ;13
.code
    .startup
    LEA SI,Vec+1
    LEA DI, Vec
    Mov CX, len -1   ;12
    CLD
    REP MOVSB
.exit
end
```

Ví dụ: Chuyển 512Byte từ vị trí 7C00 về 6000

```
.code
    .startup
        XOR AX, AX
        MOV DS, AX
        MOV ES, AX
        MOV SI, 7C00h          ; DS : SI => 7C00h
        MOV DI, 6000h          ; ES : DI => 6000h
        MOV CX, 512
        CLD
        REP MOVSB
        JMP 0: Next  ;Dat lai con tro lenh vao vi tri moi
Next:
end
```

Câu lệnh **so sánh xâu**

[REP_] **CMPSB/ CMPSW/ CMPSD**

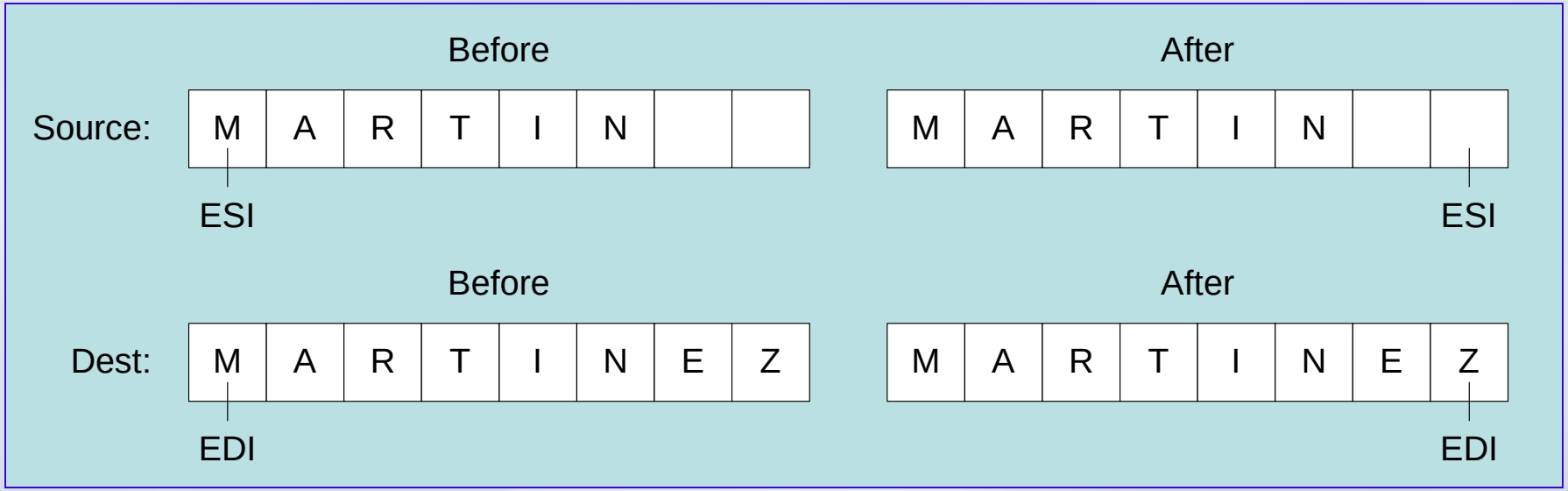
- Cập nhật thanh ghi cờ theo kết quả của phép so sánh giá trị tại DS:[SI] và ES:[DI]
  - **SI**, **DI** được tăng /giảm tự động 1/2/4 byte
- REPZ/REPE:
  - Lặp lại khi CX >0 và 2 giá trị so sánh bằng nhau
- REPNZ/REPNE
  - Lặp lại khi CX >0 và 2 giá trị không bằng nhau
- Có thể dùng lệnh nhảy có điều kiện sau đó

Câu lệnh **so sánh xâu**

Giá trị các thanh ghi SI, DI sau khi thực hiện

CLD

REPE CMPSB



Câu lệnh **SCA**n **S**tring

[REP_] **SCASB/ SCASW/ SCASD**

- Cập nhật thanh ghi cờ theo kết quả của phép so sánh giá trị AL/AX/EAX với ES:[DI]
  - **DI** được tăng /giảm tự động 1/2/4 byte
- **REPZ/REPE**:
  - Lặp lại khi CX >0 và 2 giá trị còn bằng nhau
- **REPNZ/REPNE**
  - Lặp lại khi CX >0 và 2 giá trị không bằng nhau
- **CX**: Số lần lặp tối đa
  - Kết thúc lặp, CX >0 → đ/k lặp không thỏa mãn

Câu lệnh **SCA**n **S**tring : Tìm số không đầu tiên

```
.data
   Vec DW 34, 50, 24, -57, 22, 0 , 20
   Len =($-Vec)/ TYPE Vec
.code
    .startup
    LEA DI, Vec
    Mov AX, 0
    MOV CX, Len
    REPNE SCASW ;Lặp nếu không bằng
    JNZ notExist;,Cờ zero bị xóa khi ES:[DI] – AL =0
    SUB DI, TYPE Vec
  notExist:
    .exit
end
```

Câu lệnh **STO**re **S**tring

[REP] **STOSB/ STOSW/ STOSD**

- Lưu giá trị AL/AX/EAX với ES:[DI]
  - **DI** được tăng /giảm tự động 1/2/4 byte
- Sử dụng **REP** → **CX**: Số lần lặp
- Ví dụ: Khởi tạo Vector Vec gồm 100 word

```
CLD
MOV CX, 100; 200 byte
LEA DI, Vec
XOR AX, AX; Xóa AX
REP STOSW
```

Câu lệnh **STO**re **S**tring→Ví dụ

```
.model tiny ; Ki?u b? nh?
.386                                    ;Processor 32bit
.data
        Vec DD 10000 DUP(?)
        Len1 =($-Vec)/ TYPE Vec
        Len2 = $-Vec
        Msg1 DB "Start reset 10000 DWORD ",13,10,"$"
        Msg2 DB "Start reset 40000 BYTE  ",13,10,"$"
.code
    .startup   ;?i?m b?t ??u c?a ch??ng trình
    Mov AH, 9
    LEA DX, Msg1
    Int 21h
    MOV CX,40000              ;
    MOV EAX,0FFFFFFFFh   ;
    L1:                                    ;
     PUSH CX                  ;
     LEA DI, Vec       ;
     MOV CX, Len1      ;
     REP STOSD         ;
     POP CX                ; Khôi phục CX
     LOOP L1               ; Lặp lại vòng lặp ngoài
    Mov AH, 9
    LEA DX, Msg2
    Int 21h
    MOV CX,40000          ;
    MOV EAX,0FFFFFFFFh   ;
    L2:                                    ;
     PUSH CX                  ;
      LEA DI, Vec        ;
     MOV CX, Len2       ;
      REP STOSB          ;
      POP CX             ; Khôi phục CX
     LOOP L2                  ;
     .exit                ;
End
```

Câu lệnh **LOD**e **S**tring

[REP] **LODSB/ LODSW/ LODSD**

- Copy giá trị DS:[SI] vào AL/AX/EAX
  - **SI** được tăng /giảm tự động 1/2/4 byte
- Cho phép sử dụng với **REP** nhưng vô nghĩa
  - Mỗi lần lặp, giá trị AL/AX/EAX bị thay đổi→ Sử dụng REP, AL/ AX/ EAX chứa giá trị cuối

Ví dụ → Chuyển thành xâu chữ thường

```
.model tiny    ;
.386           ;Processor 32bit
.data
  Msg DB "HeLLO WoRld !$"
  Len =($-Msg)
.code
   .startup
   LEA DI, Msg
   MOV SI, DI ; SI=DI
   Mov CX, Len
   CLD         ;clear direction
Convert:
      LODSB
      CMP AL, 'A'
      JB NotUpper
      CMP AL, 'Z'
      JA NotUpper
      OR AL, 20h
  notUpper:
      STOSB
      LOOP Convert
      Mov AH, 9
      Mov DX, OFFSET Msg
      Int 21h
   .exit
end
```

Nội dung chính

- Thanh ghi trạng thái processor
- Các lệnh sao chép dữ liệu
- Các lệnh số học và logic
- Các lệnh thao tác với bit
- Các lệnh điều khiển
- Các câu lệnh với chuỗi
- **Các câu lệnh khác**

Các lệnh vào ra: In/Out

IN	AL/AX/EAX, Port
IN	AL/AX/EAX, DX
OUT	Port, AL/AX/EAX
OUT	DX, AL/AX/EAX

- Cho phép đọc (**IN**) hoặc ghi (**OUT**) dữ liệu từ/ra một cổng
  - Port là 1 hằng số kiểu byte → 0..255
  - Sử dụng th.ghi DX khi vào /ra với các cổng 0..65535
- Để vào ra một dãy dữ liệu liên tiếp
  - [REP] **INSB/INSW/INSD**: ES:[DI] ←Port DX
  - [REP] **OUTSB/OUTSW/OUTSD**: Port DX ←DS:[SI]

Các lệnh liên quan tới chương trình con

- **CALL** Target
  - Gọi tới một chương trình con
  - **Target** là Near_Proc: 1 byte mã lệnh +2 byte vị trí
    - Thủ tục trên cùng đoạn mã. PUSH IP; IP←Proc_Ofs
  - **Target** là Far_Proc: 1 byte mã lệnh + 4 byte vị trí
    - PUSH CS; PUSH IP; IP←Proc_Ofs; CS ←Proc_Seg
  - **Target** có thể là thanh ghi, biến nhớ 16, 32 bit
- **RET** [Value]
  - Trở về từ chương trình con
  - Tùy thuộc chương trình xa/ gần mà thực hiện khôi phục CS:IP hay chỉ IP

Các lệnh liên quan ngắt mềm

- **INT** fun
  - Gọi một ngắt mềm: dịch vụ của BIOS/ DOS
  - Fun: Số hiệu ngắt (0-255)
  - Thực hiện
    - PUSHF, IF←0, IT←0, PUSH CS, PUSH IP
    - IP ← 0000:[4*fun] , CS ← 0000:[4*fun+2]
- **INTO** fun
  - Gọi ngắt nếu OF = OV=1
- **IRET**
  - Trở về từ ngắt, thực hiện các công việc
    - POP IP, POP CS, POPF

Các lệnh điều khiển vi xử lý

- Các lệnh tác động tới bit cờ
  - Carry Flag: **CLC**, **STC**, **CMC**
  - Direction Flag: **CLD**, **STD**
  - Interrupt Flag: **CLI**, **STI**
  - Trace Flag (386): **CTS/CLTS** (TF ←1)
- **HLT**
  - Dừng xử lý cho tới khi ngắt xuất hiện
- **WAIT**, **LOCK** opCode
  - Dùng trong đồng bộ các bộ vi xử lý

Bài tập: Đảo ngược xâu

```
.model tiny        ;
.386               ;Processor 32bit
.data
  Msg DB « Hello world $"
.code
   .startup
   LEA SI, Msg
   XOR CX, CX
   XOR AX, AX
L1:
   LODSB
   PUSH AX
   INC CX
   CMP AL,'$'
   JNE L1
   POP AX; Bo ky hieu '$'
   DEC CX
   LEA DI, Msg
L2:
   POP AX
   STOSB
   LOOP L2
   ;$ Van ton tai trong xau cu
   Mov AH, 9
   Mov DX, OFFSET Msg
   Int 21h
   .exit
end
```

Bài tập: Message Encryption

Key: Chuỗi độ dài 10

Msg: CHuỗi

Mã hóa bằng phép XOR

Nội dung chính

- Thành phần cơ bản của hợp ngữ
- Lập trình hợp ngữ căn bản
- Cấu trúc tập lệnh của x86
- Thủ tục và Macro
- Dữ liệu có cấu trúc
- Hợp ngữ và ngôn ngữ bậc cao

Thủ tục

```
Name PROC Type
        ;Các câu lệnh
        RET
Name ENDP
```

- **Name**:
  - Tên thủ tục do người dùng định nghĩa
- **Type**: NEAR /FAR
  - NEAR (*ngầm định*) thủ tục trong cùng đoạn
  - FAR: Thủ tục nằm ở đoạn khác với câu lệnh

Thủ tục

- **PROC** và **ENDP** là các chỉ thị, không sinh ra mã
  - Mục đích nhằm dễ đọc chương trình
- Ví dụ: 2 đoạn sau là tương đương

```
ZeroBytes:
    Xor ax, ax
    Mov cx, 128
ZeroLoop:
    mov [bx], ax
    add bx, 2
    Loop ZeroLoop
    Ret
```

```
ZeroBytes Proc
    Xor ax, ax
    Mov cx, 128
ZeroLoop:
    mov [bx], ax
    add bx, 2
    loop ZeroLoop
    Ret
ZeroBytes Endp
```

Gọi thủ tục

```
CALL [Near/far Ptr] Name    ; gọi trực tiếp
CALL [Near/far Ptr] Address ; gọi gián tiếp
```

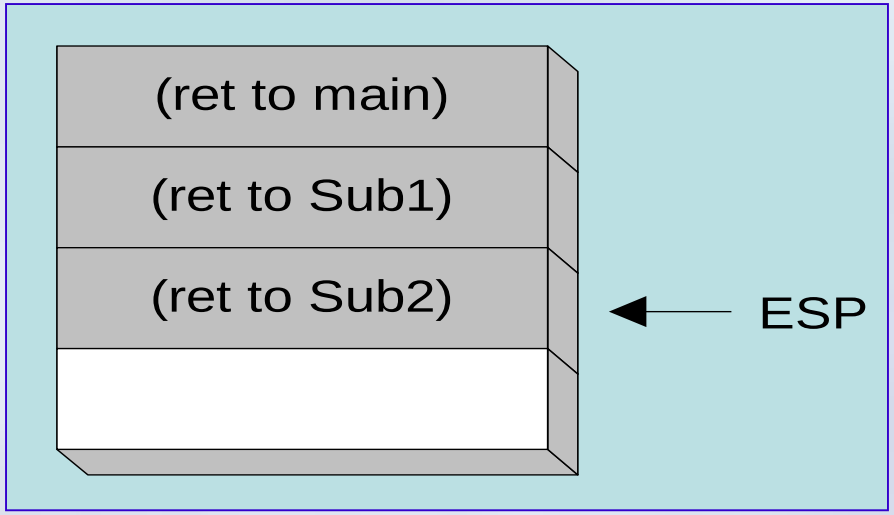
- **Address** có thể là
  - Biến nhớ 16 bit/32 bit
  - Thanh ghi 16 bít
- **Far Ptr**
  - Sinh ra lời gọi xa (lưu CS và IP vào Stack)
- **RET** [Value]
  - Điểm kết thúc của một chương trình con
  - Khôi phục con trỏ lệnh và SP ← SP + Value

Ví dụ

```
.model tiny              ; Kiểu bộ nhớ
.386                     ;Processor 32bit
.data
        Msg DB "Hello world $"
.code
    .startup             ;Điểm bắt đầu của chương trình
        LEA DX, Msg
        CALL printf
     .exit               ;Điểm kết thúc
    printf PROC          ;Phần khai báo các thủ tục
        MOV AH,9
        INT 21h
        RET
    printf ENDP
End
```

Gọi liên tiếp các thủ tục

Khi thủ tục Sub3 được goi,
Stack đã chứa địa chỉ trở về của
3 thủ tục trước đó




```
main PROC
    .
    .
    call Sub1
    exit
main ENDP

Sub1 PROC
    .
    .
    call Sub2
    ret
Sub1 ENDP

Sub2 PROC
    .
    .
    call Sub3
    ret
Sub2 ENDP

Sub3 PROC
    .
    .
    ret
Sub3 ENDP
```

Lưu lại trạng thái vi xử lý

- Giả thiết
  - Putc: in ra 1 ký tự trong AL
  - Putcr: in ra dấu xuống dòng
- Dự kiến thực hiện
  - Gọi thủ tục Print và putcr 10 lần
  - Thủ tục Print in ra 40 ký tự trắng
- Thực tế thực hiện
  - Print và putcr bị gọi lặp vô hạn
  - Lý do:
    - kết thúc Print cx= 0
    - Loop L0 giảm CX →CX=0FFFF
- Cần lưu các th.ghi bị thay đổi
  - Do chương trình gọi lưu
  - Do chương trình bị gọi lưu

```
    Mov CX, 10
L0:
    call Print
    call putcr
    Loop L0
.
Print Proc near
    mov al, 32
    mov cx, 40
    L1:
        call putc
        Loop L1
    Ret
Print Endp
```

Tham số của thủ tục

- Các kiểu tham số
  - Giá trị
  - Địa chỉ
- Vị trí đặt tham số
  - Thanh ghi
  - Biến toàn cục
  - Stack
- Trả về kết quả (hàm)
  - Trong thanh ghi
  - Stack

Truyền tham số

- Sử dụng thanh ghi
  - Khi thủ tục yêu cầu ít tham số
  - Quy ước
    - Thứ tự sử dụng thanh ghi :AX, DX, SI, DI, BX, CX
    - Khi truyền theo địa chỉ
      - SI, DI, BX chứa offset
      - DS:SI, DS:DI, DS:BX, chứa địa chỉ
  - Chú ý
    - Nên ghi lại các thanh ghi bị thay đổi

Ví dụ: WriteDec

```
writeDec PROC
    ;-----------------------------------------
    ;Ghi ra so nguyen he co so 10,
   ;Tham so cho trong thanh ghi EAX
    ;-----------------------------------------
      Mov BX, 10          ;
      XOR CX, CX          ;Xóa CX
   divLoop:
      XOR EDX, EDX        ;EDX=0
      DIV EBX             ;EDX:EAX/EBX
      PUSH DX             ;Cat so du
      INC CX              ;
      CMP EAX, 0          ;KQ=0 ?
      JNZ divLoop
```

```
printLoop:
    POP AX          ; So du <10
    ADD AX,30h     ;Chuyen ASCII
    MOV AH,0Eh     ;
    XOR BX,BX      ;
    INT 10h         ;
    LOOP printLoop
    RET
writeDec ENDP
```

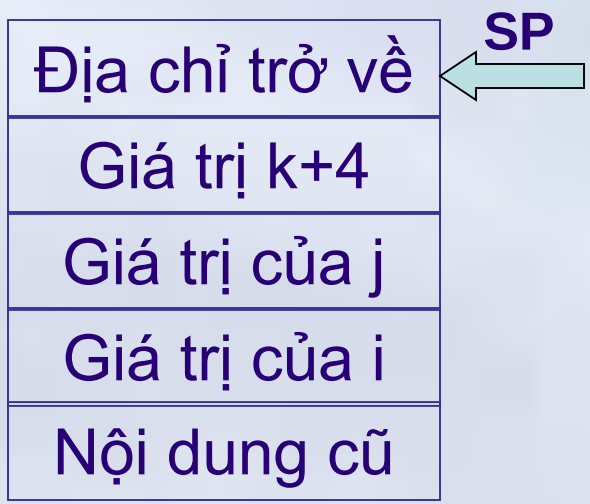
Truyền tham số

- Sử dụng các biến toàn cục
  - Khi có nhiều tham số hoặc kích thước lớn
  - Dùng một khối nhớ để chứa các tham số
  - Không hiệu quả và khó cho thủ tục đệ quy
- Sử dụng Stack
  - Cách thực hiện của ngôn ngữ cấp cao
  - Đặt tham số vào stack trước khi gọi thủ tục
  - Trong thủ tục, lấy tham số từ stack
  - Chú ý: đỉnh Stack là địa chỉ quay trở lại (2/4 byte)
    → Cần phải xử lý thích hợp

Truyền tham số sử dụng stack

```
Routine(i, j, k + 4)
```

```
Push i
Push j
Mov ax, k
Add ax, 4
Push ax
Call Routine
```



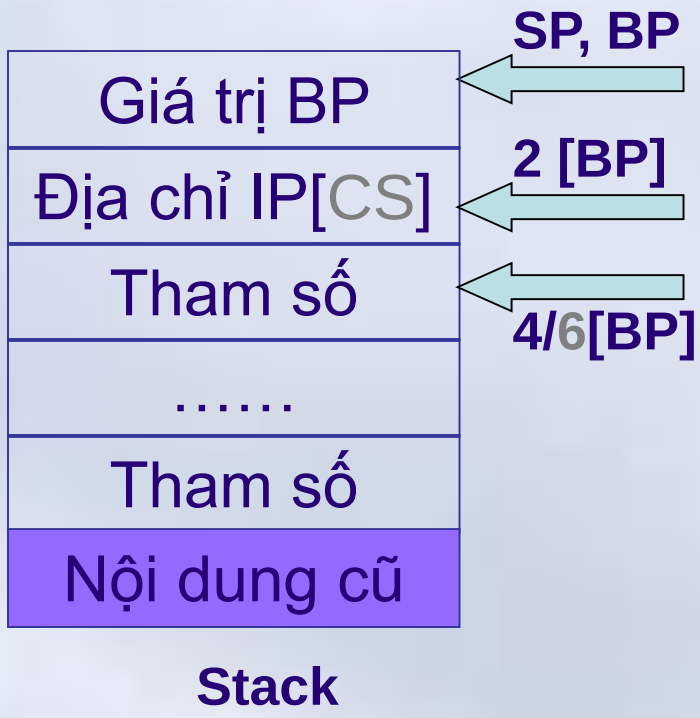

```
Routine PROC  near ;FAR
    Pop RtnAdrs; 2/4 byte
    Pop kParm
    Pop jParm
    Pop iParm
    Push RtnAdrs
    .
    .
    RET
Routine ENDP
```

Truyền tham số sử dụng Stack

- Sử dụng thanh ghi **BP**: Base Pointer
  - Nếu **BP** trỏ tới địa chỉ **Addr** ⇒ **N[BP]** trỏ tới **Addr + N**
- Sử dụng **BP** để chỉ đánh chỉ số tham số trong Stack
- Chương trình con chuẩn

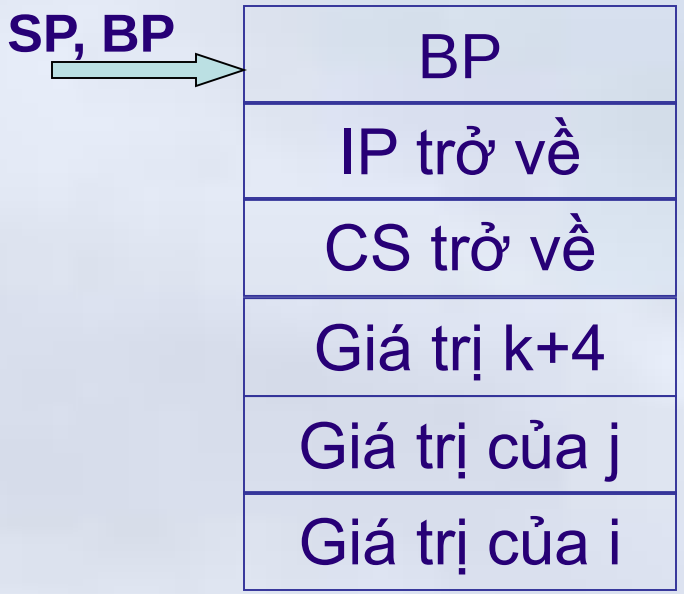
```
StdProc PROC NEAR;  FAR
        PUSH BP
        MOV BP, SP
        ;Sử dụng tham số 4[BP]

        POP BP
        RET ParSize;
StdProc END
;ParSize: số byte tham số
```



Truyền tham số sử dụng Stack

```
Push i
Push j
Mov ax, k
Add ax, 4
Push ax
Call Routine
```




```
Routine PROC  FAR
    PUSH BP
    MOV BP, SP
    MOV kPam, 6[BP]
    MOV jPam, 8[BP]
    MOV iPam, 10[BP]
    ;Xử lý
    ;
    POP BP
    RET 6; 3 tham số
Routine ENDP
```

Truyền tham số dùng Stack →Truyền theo biến

```
void  subProc(int * a; int b; int c)
{
        *a = b + c;
}
```

```
void  subProc(&a, 3, 4)
```

```
;a là biến ở xa
;a biến gần, có thể bỏ lệnh đầu
PUSH SEG a ;Đặt địa chỉ của a
PUSH OFFSET a     ; lên stack.
PUSH 3 ; Tham số thứ 2 và 3
PUSH 4 ; truyền theo trị.
CALL subProc;
```

```
subProc PROC NEAR
    PUSH BP     ;Lưu BP
    MOV BP, SP
    PUSH ES     ;Lưu các thanh
    PUSH AX     ;ghi bị thay đổi
    PUSH BX     ; trong thủ tục
    LES BX, 8[BP]; ES:BX trỏ tới a
    MOV AX, 6[BP] ; Tham số b
    ADD AX, 4[BP] ; Tham số c
    MOV ES:[BX], AX ;
    POP BX      ;Khôi phục thanh
    POP AX      ;ghi bị thay đổi
    POP ES      ;Trong thủ tục
    POP BP      ; Khôi phục BP
    RET 8       ;8 byte tham số
subProc ENDP
```

Hàm→ Đòi hỏi trả về một kết quả

- Trả về trong thanh ghi
  - Giá trị: word : AX, DX, CX, SI, DI, BX
    dword: DX:AX, EAX, EDX, ECX,..
  - Offset: BX, SI, DI, DX, EBX, ESI, EDI, EDX
  - Con trỏ:ES:DI, ES:BX, DX:AX, ES:SI; kg s/d DS
- Đặt trong một vị trí nhớ
  - Trả lại kết quả tại một biến toàn cục
  - Trả lại địa chỉ (*dùng cặp thanh ghi*) của một khối tham số chứa kết quả
    - Ví dụ: Xin bộ nhớ để trả về một cấu trúc, chương trình gọi có trách nhiệm giải phóng vùng nhớ này

Hàm→Trả lại kết quả trong Stack

- Đặt vào stack một vài giá trị nháp để tạo không gian để chứa kết quả
- Trước khi kết thúc hàm, lưu kết quả tại vùng nhớ
- Chương trình gọi lấy kết quả từ stack
- Ví dụ

```
int  subProc(int b; int c) {
            return b + c;
}

A = subProc(5, x);
```

Hàm→Trả lại kết quả trong Stack

```
;CALLER
PUSH AX; Lấy chỗ
PUSH 5
PUSH Y
CALL subProc;
POP AX
MOV  [A], AX
```

```
subProc PROC NEAR
    PUSH BP  ;Lưu BP
    MOV BP, SP
    PUSH AX  ;ghi bị thay đổi
    MOV AX, 6[BP]
    ADD AX, 4[BP]
    MOV 8[BP],AX
    POP AX    ;
    POP BP    ; Khôi phục BP
    RET 4     ;6 byte tham số
subProc ENDP
```

Đệ quy

- Khi thủ tục gọi (trực tiếp, gián tiếp) đến chính nó
  - Thủ tục đệ quy cần có điều kiện dừng

```
;Đệ quy tới khi AX=0
;Sử dụng stack
Recursive Proc
        DEC AX
        JZ QuitRecurse
        CALL Recursive
   QuitRecurse:
        RET
Recursive ENDP
```

```
;Đệ quy tới khi AX=0
;Không dùng Stack, không có
;các lệnh Call, RET →Nhanh
Recursive Proc
RepeatAgain:
    DEC AX
    JNZ RepeatAgain
    RET
Recursive ENDP
```

Đệ quy→Tính giai thừa

```
factorial PROC
    ;-----------------------------------------------------------------------------
    ;Tham so dau vao trong thanh ghi CL, Tra ve cho trong thanh ghi EAX
    ;-----------------------------------------------------------------------------
      CMP CL, 0
      JE stopRecursive
      PUSH ECX
      DEC CL
      Call factorial
      POP ECX
      MUL ECX
      RET
      stopRecursive:
          MOV EAX, 1
          RET
   factorial ENDP
```

Ví dụ: Hàm sinh số ngẫu nhiên

Sử dụng hàm sinh số

$$Xn = 65X_{n-1} + 13 \%2^{16}$$

```
;Biến lưu giá trị đầu
Seed DW 0
```

```
;Hàm khởi tạo giá trị đầu
;Tham số cho trong AX
initSeed PROC
    MOV Seed, AX
    RET
initSeed ENDP
```

```
;Kết quả trả về trong AX
nextInt Proc
    PUSH EDX
    MOVZX EAX, Seed
    MOV EDX, EAX
    SHL EAX, 6
    ADD EAX, EDX
    ADD EAX, 13
    MOV Seed, AX
    POP EDX
    RET
    nextInt EndP
```

Sắp xếp dãy số

```
bubleSort Proc
;Tham so duoc truyen theo Stack
;Địa chỉ và so phan tu cua mang
    PUSH BP
     MOV BP, SP
     MOV BX, [BP+6]  ; Địa chỉ
     MOV CX, [BP+4]  ; Số ptu
     DEC CX
          ……..……..
     POP BP
    RET 4
    bubleSort EndP
```

```
L1:
        PUSH CX
        MOV SI, BX
L2:
        MOV AX, [SI]
        CMP AX, [SI+2]
        JB L3
        XCHG AX, [SI+2]
        MOV  [SI],AX
L3:
        ADD SI, 2
        LOOP L2
        POP CX
        LOOP L1
```

Modul hóa chương trình

Khi chương trình lớn ⇒ Nhiều file nguồn

1. Chỉ thị **INCLUDE <Name>**
   - Chèn file «Name» vào văn bản đang soạn thảo
     - Thường dùng để đưa vào các hằng, macro, khai báo thủ tục bên ngoài..
2. Tạo các Modul riêng
   - Dễ dàng phát triển, viết mã, gỡ rối..
     - Chỉ phải dịch lại phần code đã thay đổi
     - Cho phép che giấu các thủ tục, biến khi cần
   - Thực hiện
     - Các Modul được hợp dịch riêng thành các *.obj
     - Các file *.Obj được liên kết lại thành file thực thi

Modul hóa chương trình→ Chỉ thị INCLUDE

Main.asm

```
.model tiny
.386      ;Processor 32bit
.code
    .startup
          MOV EAX,12345
           Call writeDec
           Call writeLn
     .exit;
Include Lib.asm
End
```

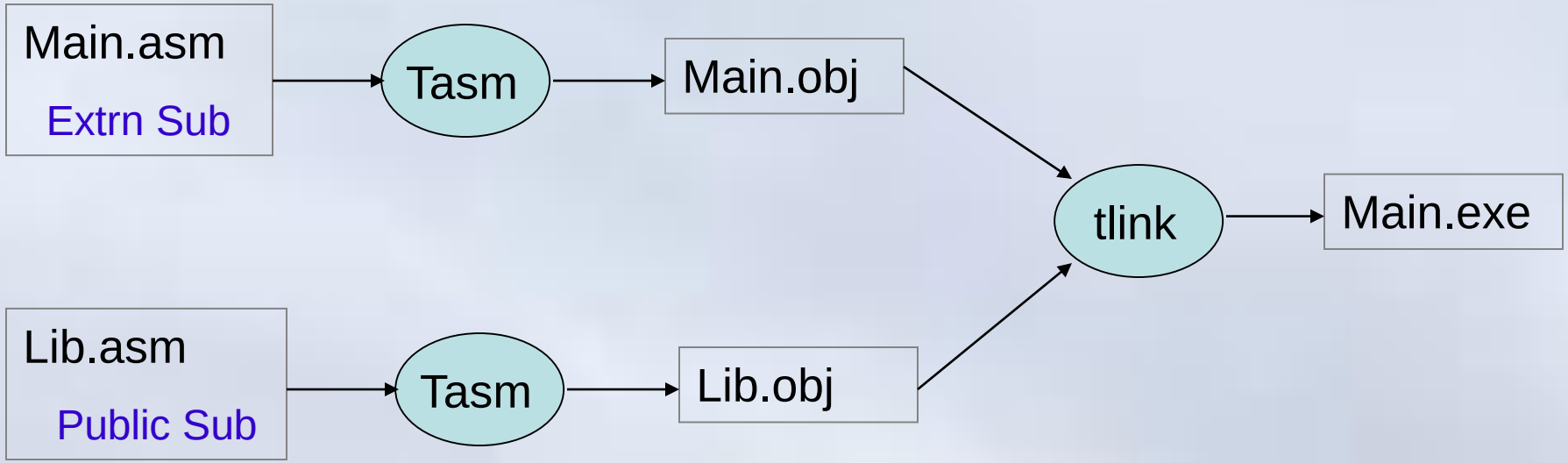
Lib.asm

```
    writeDec PROC
          ;.............
          RET
    writeDec ENDP
    writeLn PROC
          RET
    writeLn ENDP
```

```
.model tiny
.386      ;Processor 32bit
.code
    .startup
          MOV EAX,12345
          Call writeDec
          Call writeLn
     .exit;
    writeDec PROC
          ;.............
          RET
    writeDec ENDP
    writeLn PROC
          RET
    writeLn ENDP
End
```

Modul hóa chương trình →Tạo các Modul

- Chỉ thị **EXTRN** <Label>: type[,<Label>:type]
  - Nhãn Label được khai báo bên ngoài
- Chỉ thị **PUBLIC** <Label> [,<Label]
  - Nhãn Label có thể nhìn thấy từ modul khác



Modul hóa chương trình →Tạo các Modul

```
.model tiny
.386                ;Processor
   32bit
.data
.code
extrn writeDec :near
extrn writeLn   :near
    .startup
         MOV EAX,12345
           Call writeDec
           Call writeLn

En
```

```
.model Tiny
.386
.code
    PUBLIC writeDec
    PUBLIC writeLn
    writeDec PROC
          ;.....................
          RET
    writeDec ENDP

    writeLn PROC
```

```
C:\...\>tasm Lib
C:\...\>tasm Main
C:\...\>tlink Main+Lib /t/3
```

Macro

- Tên của một khối lệnh hợp ngữ
- Có thể được gọi như thủ tục (không s/d CALL)
  - Khi được gọi, trình hợp dịch sẽ thay thế Macro bằng các lệnh được Macro định nghĩa
- Macro khác thủ tục
  - Không sinh ra mã lệnh **CALL** và **RET**
  - Thay Macro trực tiếp bằng mã lệnh
    - Chương trình đích có kích thước lớn hơn
- Macro phải được định nghĩa trước khi dùng

Định nghĩa Macro

Name **MACRO** [Param1, Param2,..]
    statement-list
ENDM *Name*

Gọi Macro

- Sử dụng trực tiếp tên macro
- Số tham số phù hợp với số tham số khai báo
  - Các tham số sẽ được thay bởi các tham số truyền vào tương ứng khi thay thế nội dung Macro
  - Tham số được xử lý như text bởi bộ tiền xử lý
- Khi triển khai Macro, sẽ thay bằng mã hợp ngữ

Macro→Ví dụ

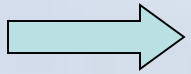
```
writeln MACRO
    MOV AX,0E0Ah ;
    INT 10h
    MOV AX,0E0Dh
    INT 10h
ENDM writeln
.model tiny
.data
.code
    .startup
    writeln
    writeln
    .exit
End
```

```
.model tiny
.data
.code
    .startup
    MOV AX,0E0Ah ;
    INT 10h
    MOV AX,0E0Dh
    INT 10h
    MOV AX,0E0Ah ;
    INT 10h
    MOV AX,0E0Dh
    INT 10h
    .exit
End
```

Macro→Ví dụ

```
putc MACRO Ch
    MOV AH,0Eh
    MOV AL, Ch
    INT 10h
ENDM
```

```
.code
    putc 'A'
    putc 'B'
    putc 67
end
```



```
.code
    MOV AH,0E
    MOV AL, 'A'
    INT 10h
    MOV AH,0E
    MOV AL, 'B'
    INT 10h
    MOV AH,0E
    MOV AL, 67
    INT 10h
End
```

Putc 'A'

Putc 'B'

Putc 67

Macro→Sự trùng lặp

- Trong Macro có sử dụng nhãn lệnh
  - Gọi Macro nhiều lần ⇒ nhãn bị lặp lại
- Sử dụng chỉ thị **LOCAL**
  - Phải xuất hiện ngay sau chỉ thị khai báo Macro

```
toLower MACRO Char
    MOV AL, Char
    CMP AL, 'A'
     JB NotUpper
     CMP AL, 'Z'
     JA NotUpper
     OR AL, 20h
   notUpper:
ENDM toLower
```

```
toLower MACRO Char
    Local notUper
    MOV AL, Char
    CMP AL, 'A'
     JB NotUpper
     CMP AL, 'Z'
     JA NotUpper
     OR AL, 20h
   notUpper:
ENDM toLower
```

Bài tập

- Viết một hàm cho phép nhập từ bàn phím một xâu ký tự (có soạn thảo đơn giản)
- Viết một hàm cho phép nhập từ bàn phím một biểu thức đơn giản (+, *). Trả về kết quả của biểu thức

Nội dung chính

- Thành phần cơ bản của hợp ngữ
- Lập trình hợp ngữ căn bản
- Cấu trúc tập lệnh của x86
- Thủ tục và Macro
- Dữ liệu có cấu trúc
- Hợp ngữ và ngôn ngữ bậc cao

Giới thiệu

- Kiểu mảng
  - Dãy các Byte/Word/.. nhớ liên tiếp nhau
  - Khai báo → Sử dụng toán tử **DUP**
  - Ví dụ: Vector W 100 DUP (?)
- Kiểu chuỗi
  - Mảng các ký tự; kết thúc bởi ký tự đặc biệt
- Kiểu con trỏ
  - Địa chỉ của một đối tượng trong bộ nhớ
- Kiểu cấu trúc
  - Gồm nhiều thành phần kết hợp với nhau

Kiểu cấu trúc

Khai báo và sử dụng

```
PosType STRUC
        Row  DW  ?
        Col   DW ?
  PosType ENDS
.data
        Point PosType ?
.Code
        MOV Point.Row, AX
        MOV [Point.Col], BX
```

Nội dung chính

- Thành phần cơ bản của hợp ngữ
- Lập trình hợp ngữ căn bản
- Cấu trúc tập lệnh của x86
- Thủ tục và Macro
- Dữ liệu có cấu trúc
- Hợp ngữ và ngôn ngữ bậc cao

Giới thiệu

- Ngôn ngữ bậc cao
  - Sử dụng trong phát triển ứng dụng
  - Giải phóng lập trình viên khỏi chi tiết mức thấp
- Hợp ngữ
  - Điều khiển trực tiếp và hoàn toàn phần cứng
  - Tốc độ cao, kích thước gọn
  - Sử dụng trong viết driver thiết bị, hệ điều hành, chương trình nhúng,..
- Kết hợp ngôn ngữ bậc cao và hợp ngữ
  - Mở rộng khả năng của ngôn ngữ bậc cao
  - Giúp tăng tốc cho những phần đặc biệt

Chèn chỉ thị ASM vào ngôn ngữ bậc cao (TC)

- Chèn mã hợp ngữ vào ngôn ngữ bậc cao
- Thực hiện nhanh do không dùng CALL, RET
- Đơn giản do không sử dụng khái niệm đối tượng bên ngoài (**extern**)
- Cú pháp

```
asm statement
asm{
    statement //chú thích
    ……
    statement
}
```

Ví dụ

```
void clrscr(){
  asm{
    MOV AX,0600h
    MOV CX,0
    MOV BH,7
    MOV DX,184Fh
    int 10h       //Xoa màn hình
    MOV AH, 2
    MOV DX, 0
    MOV BH, 0
    int 10h   //Dat vi tri con tro
  }
}
```

```
void printf(char * str){
    asm{
        MOV AH, 9
        MOV DX, str
        int 21h
    }
}
int main(){
    char str[] = "Hello world$";
    clrscr();
    printf(str);
    return 0;
}
```

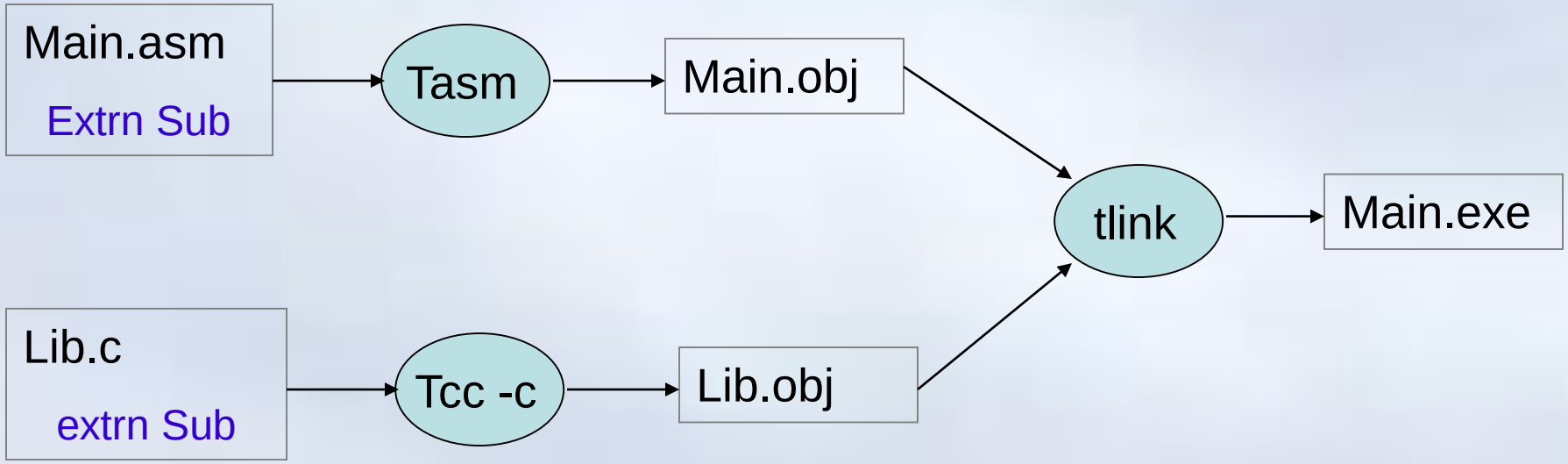
Ví dụ 2 : Nhãn lệnh và dữ liệu

```
void strstd(char * str){
  char eos ='$';
    asm{
       PUSH SI
       MOV SI, str
       L:
          LODSB
          cmp AL,0
          JNZ L
          MOV AL, eos
          MOV BYTE PTR [SI-1], eos
       POP SI
    }
}
```

Loi: undefined label L (!?)

```
void strstd(char * str){
   char eos ='$';
    asm{
       PUSH SI
       MOV SI, str
    }
    L:
     asm{
          LODSB
          cmp AL,0
          JNZ L
          MOV AL, eos
          MOV BYTE PTR [SI-1], AL
       POP SI
    }
}
```

Liên kết giữa các modul



- Từ C gọi hàm của hợp ngữ
- Từ hợp ngữ gọi hàm của C
  - Chú ý quy ước gọi hàm và quy cách truyền tham số

Ví dụ

```
PUBLIC _cls       ; Quy ước của C
PUBLIC _writeln   ; Tên bên ngoài
PUBLIC _writestr  ; bắt đầu bởi '_'
_writeLn PROC
        ;...............
    RET
_writeLn ENDP
_writeStr PROC
    PUSH BP
    MOV BP, SP
    MOV AH, 9
    MOV DX, [BP+4]
    Int 21h ; Xau ket thuc boi '$'
    POP BP
    RET 2;// CÓ 1 th
_writeStr ENDP
```

```
extern void writeLn();
extern void cls();
extern void writeStr();
int main(){
    char str[] = "Hello$";
    cls();
    writeLn();
    writeStr(str);
    return 0;
}
```

```
C:\...\>tcc -c -I\Tc\Include test.c
C:\...\>tlink \Tc\Lib\c0s+test+lib,test, , \Tc\Lib\cs.lib
```