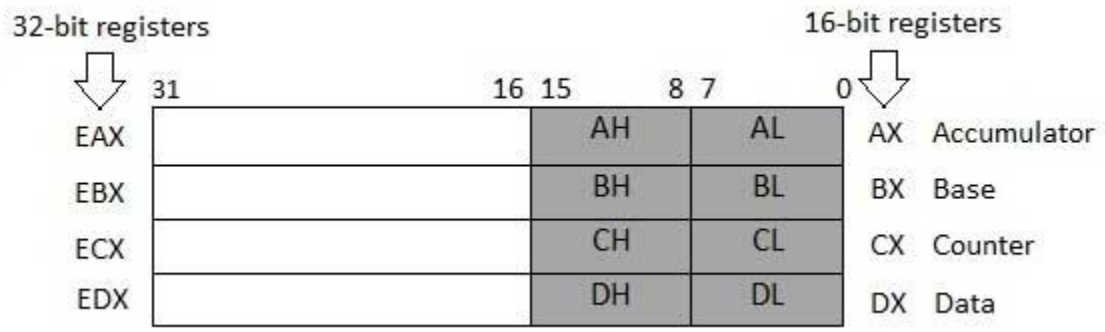
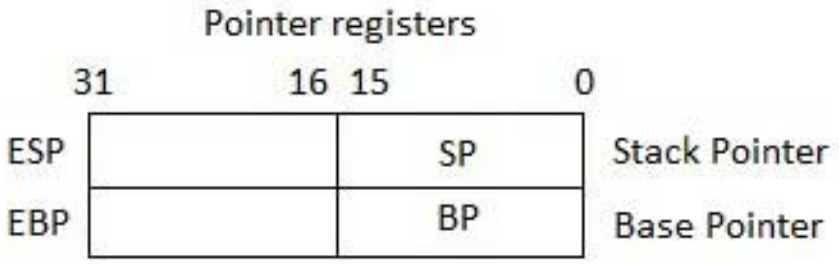
Báo cáo lý thuyết 2 tuần

1. Thanh ghi đa năng (General Registers)
   - Thanh ghi dữ liệu (Data Registers)
   - Thanh ghi con trỏ (Pointer Registers)
   - Thanh ghi chỉ số (Index Registers)

1.1 Thanh ghi dữ liệu

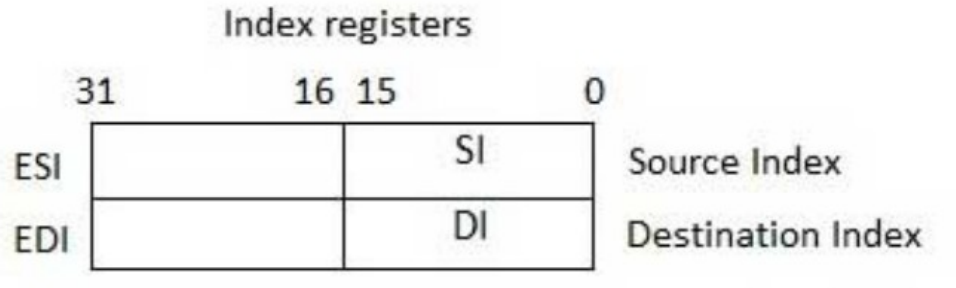


1.2 Thanh ghi con trỏ



ESP (Stack Pointer) là thanh ghi con trỏ stack trỏ tới đỉnh của stack.
EBP (Base Pointer) là thanh ghi con trỏ cơ sở, dùng để tham chiếu đến các biến tham số trong chương trình con.
EIP (Instruction Pointer) là thanh ghi con trỏ lệnh, trỏ tới địa chỉ của lệnh thực thi tiếp theo.

1.3 Thanh ghi chỉ số



Thanh ghi chỉ số (Index Register) thường được dùng để đánh chỉ số cho các địa chỉ của mảng, xâu.

ESI (Source Index) được sử dụng làm địa chỉ nguồn cho các phép toán với xâu.

EDI (Destination Index) được sử dụng làm địa chỉ đích cho các phép toán với xâu.

2. Thanh ghi cờ (Flags Registers)

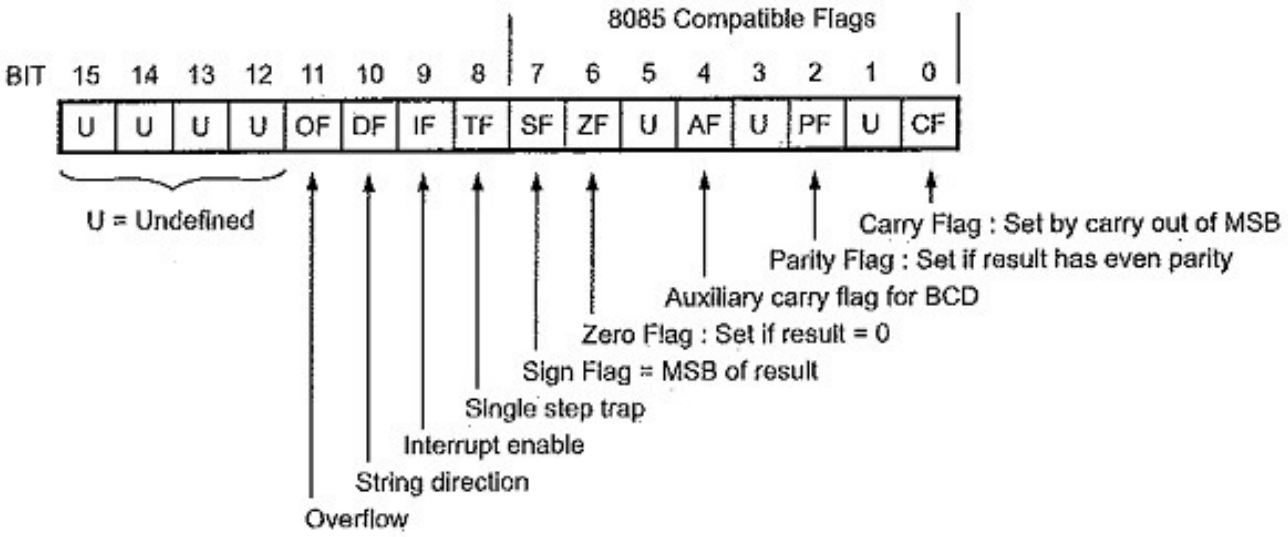


Fig. 6.6 8086 Flag register bit pattern

Các loại CPU Flag: Overflow, Direction, Trap, Sign, Zero, Auxiliary, Parity, Carry

- Overflow Flag - OF: được đặt khi một chỉ thị (instruction) sinh ra kết quả vượt ra ngoài khoảng của địa chỉ đích..
  Overflow Flag xảy ra trong trường hợp:
  - quá lớn, vượt qua giới hạn trên nếu là số dương
  - quá nhỏ, vượt qua giới hạn dưới nếu là số âm

- Direction Flag - DF: xác định hướng duyệt chuỗi.
  - DF = 1: duyệt từ phải → trái
  - DF = 0: duyệt từ trái → phải

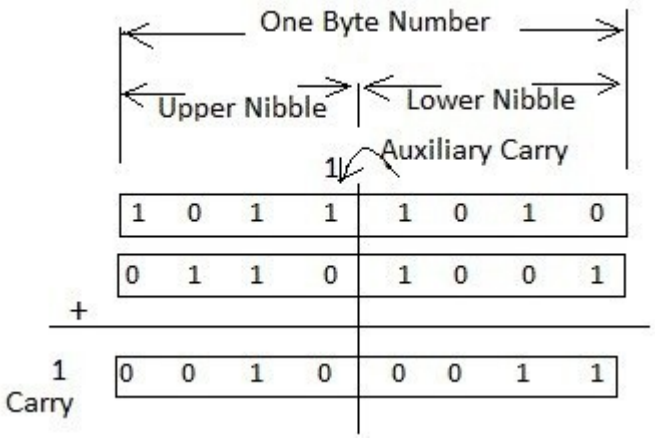
- Interrupt Flag - IF: quyết định việc thực hiện tín hiệu ngắt
  - IF = 1: thực hiện ngắt
  - IF = 0: bỏ qua ngắt

- Trap Flag - TF: đặt trap sau mỗi instruction để vào chế độ single-step, cho phép thực hiện từng lệnh một (như trong khi debug).

- Sign Flag: được đặt theo bit cao nhất của toán hạng đích của phép tính.
  - SF = 1 nếu kết quả toán hạng là âm
  - SF = 0 nếu kết quả toán hạng là dương

- Zero Flag: được đặt khi kết quả của phép tính bằng 0 hoặc phép so sánh bằng nhau.

- Auxiliary Carry Flag: chứa giá trị nhớ khi chuyển từ bit có trọng số 3 lên bit có trọng số 4 (nhớ từ lower nibble sang high nibble) khi thực hiện phép toán số học.



- Carry Flag: chứa giá trị nhớ của MSB (most significant bit hay the bit furthest to the left)
  - mov al, -1
    add al, 1

  → al = 0, ZF = 1, CF = 1
  - mov al, 6
    sub al, 9

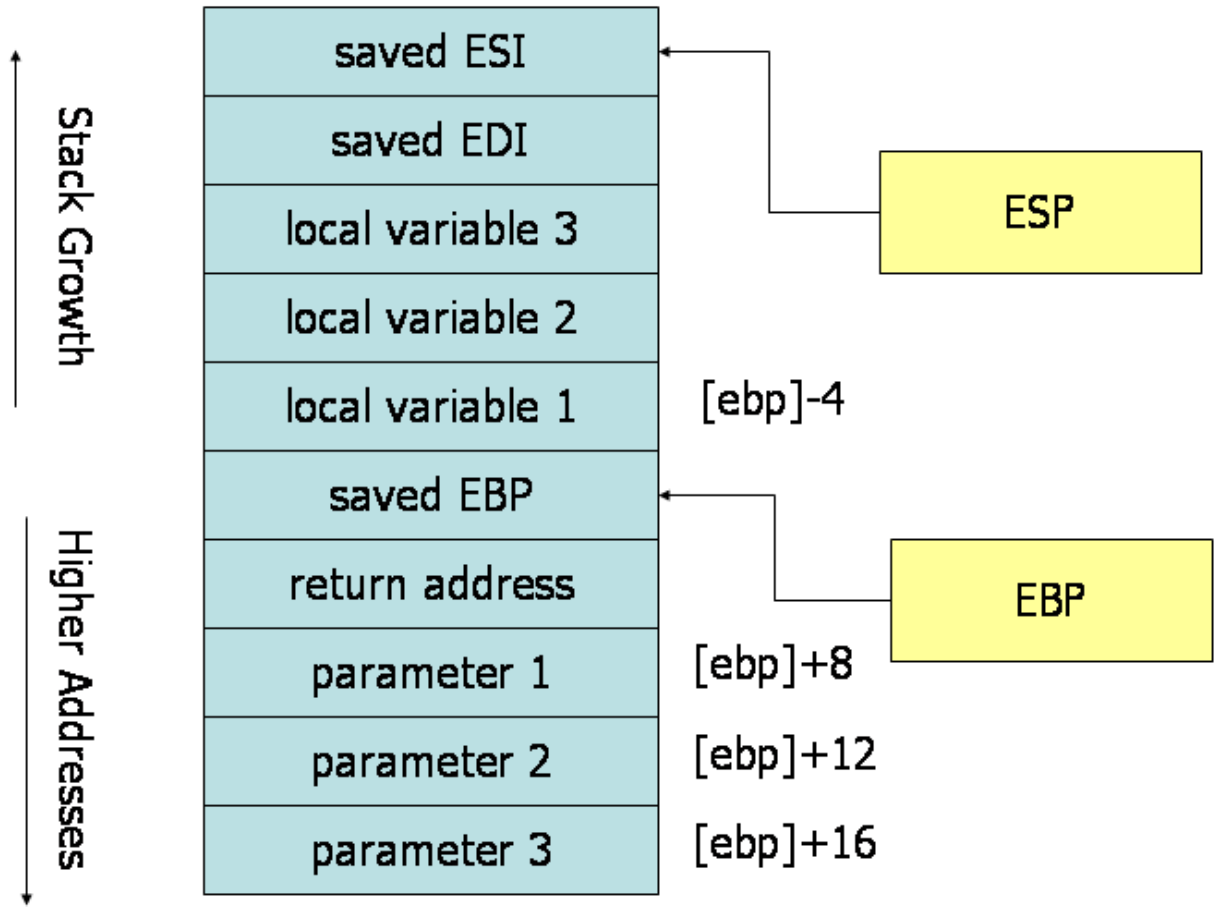
  → al = -3, SF = 1, CF = 1

- Parity Flag: xác định số lượng chẵn hoặc lẻ của số bit 1 trong kết quả phép toán
  - PF = 1 nếu số lượng bit 1 là lẻ
  - PF = 0 nếu số lượng bit 1 là chẵn

→ DF, IF, TF là cờ điều khiển
OF, SF, ZF, CF, AF và PF là cờ trạng thái

3. Calling convention

Để đơn giản và tách biệt các chương trình con, các lập trình viên đã đề ra một quy tắc chung ‘calling convention’. “Calling convention” là một giao thức về cách ‘call’ và ‘return’ từ một chương trình con.



Hình minh họa cho nội dung của stack trong khi thực hiện 1 chương trình con với 3 tham số (parameters) và 3 biến cục bộ (local variables). Hệ thống mô phỏng trên cấu trúc 32 bit.

Mô tả:

Tham số 1 đặt tại offset +8 bytes tính từ base pointer EBP. Phía trên các tham số và dưới base pointer là return address được đặt bởi hàm ‘call’. Khi hàm ‘ret’ được gọi bởi chương trình con, hàm ‘ret’ sẽ nhảy đến địa chỉ return address lưu trên stack.

Quy tắc ‘call’:

- Lưu lại các caller-saved register (EAX, EBX, ECX, EDX) nếu sau khi ‘call’ không muốn nội dung các register bị thay đổi. Caller push các caller-saved registers này vào stack trước khi ‘call’ và pop lại sau khi chương trình con ‘ret’.
- Để truyền tham số cho chương trình con, push chúng lên stack trước khi ‘call’. Các tham số nên truyền theo thứ tự đảo ngược (tham số cuối thì truyền trước). Khi stack đi xuống, tham số nhỏ hơn ở địa chỉ nhỏ hơn nên sẽ được thực hiện trước.

- Khi gọi hàm ‘call’, hàm sẽ đặt return address lên trên cùng của tham số và dẫn tới lệnh của chương trình con. Chương trình con nên tuân theo ‘Callee rules’.

Để khôi phục trạng thái cũ, caller nên:

- Xóa tham số trên stack
- Khôi phục giá trị các caller-saved register bằng cách push và pop chúng trên stack.

Callee Rules:

1. Push EBP lên stack, sau đó sao chép giá trị ESP vào EBP:
   ```
   push ebp
   mov ebp, esp
   ```
2. Base pointer EBP trên stack được dùng như một mốc để tìm tham số và biến trên stack. Khi chương trình con được thực thi, base pointer EBP chứa bản sao của giá trị stack pointer. Các tham số và biến sẽ luôn ở một vị trí xác định tính từ base pointer EBP (lúc này đang trên stack). Sau khi thực hiện chương trình con, caller có thể khôi phục EBP bằng cách pop EBP.
3. Cấp phát không gian trên stack cho biến. Khi stack đi xuống (đi đến nơi có địa chỉ cao hơn), các biến sẽ được truy cập lần lượt từ trên xuống. 3 biến (mỗi biến 4 bytes) nên cần cấp 12 bytes → Giảm 12 bytes trên stack.
   ```
   sub esp, 12
   ```
4. Lưu các callee-saved register bằng cách push lên stack
5. Sau khi thực thi chương trình con, khôi phục các callee-saved registrer
6. Khôi phục lại khoảng nhớ đã cấp
   ```
   mov esp, ebp
   ```
7. Khôi phục base pointer
   ```
   pop ebp
   ```
8. Gọi hàm ‘ret’