



BÁO CÁO THỰC TẬP BKAV-2016

Đề tài: Lập trình bằng hợp ngữ Assembly 32 bit và tìm hiểu về các thanh ghi trong vi xử lý Intel 80386

Sinh viên thực hiện: Nguyễn Hữu Phú

Trường: ĐHBKHN

Người hướng dẫn: Tạ Đức Thiện and Nguyễn Đức Mạnh

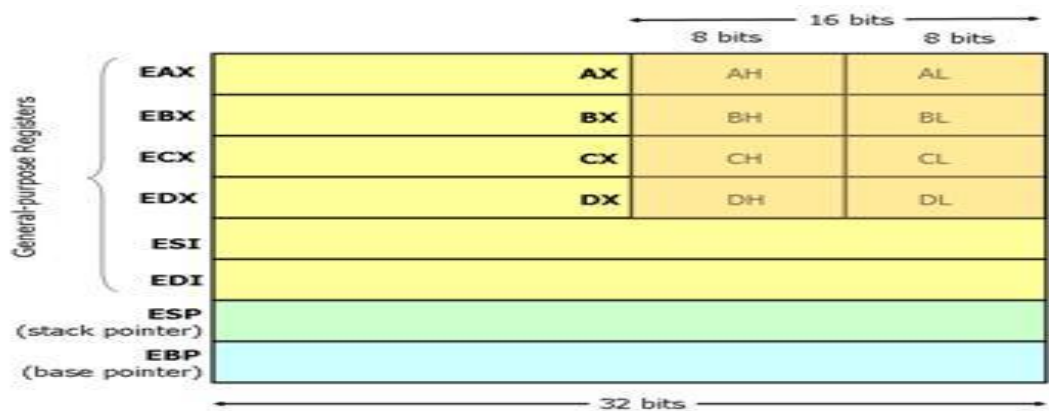
Hà Nội 7/2016

I.Chức năng và cấu tạo của các thanh ghi trong vi xử lý Intel 80386

-Trong vi xử lý họ Intel 80386 gồm có các thanh ghi 16 bit,8 bit là kế thừa của họ Intel 80x86



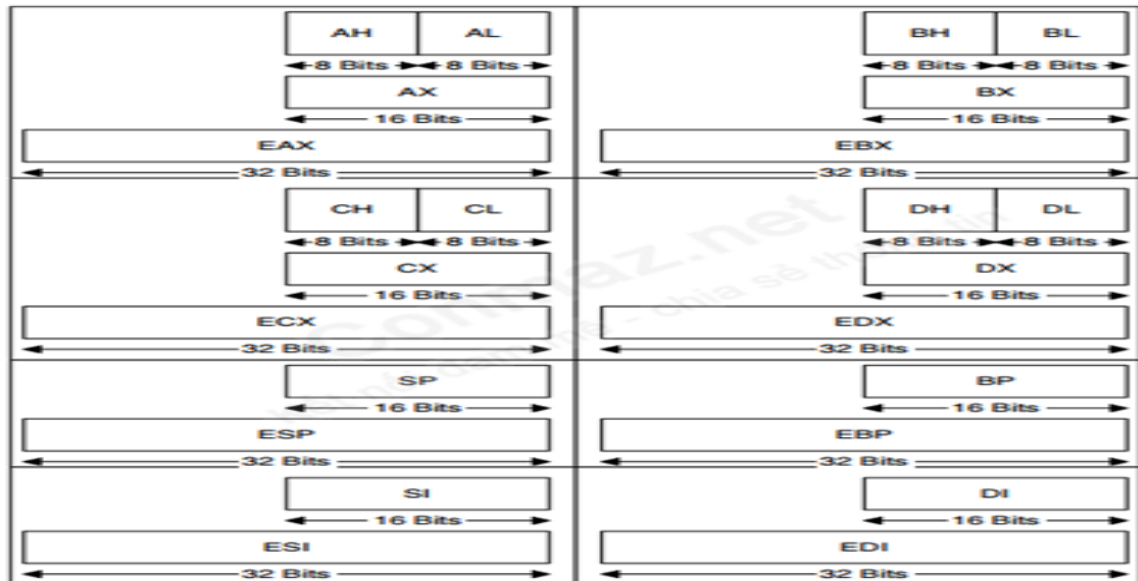
.Ngoài ra Intel 80386 có các thanh ghi 32 bit có độ dài từ nhớ 32 bit, không gian địa chỉ 32 bit có thể quản lý tối đa 4GB không gian nhớ



-Có 4 nhóm các thanh ghi:

+nhóm 1: gồm thanh ghi đa năng 32 bit ,16bit,8 bit

- 32 bit: **eax,ebx,ecx,edx** các thanh ghi này thực chất là sự mở rộng của các thanh ghi 16 bit.
- 16 bit:AX,BX,CX,DX
- 8bit:AL,AH,BL,BH,CH,CL,DL,DH .Các thanh ghi này là phần byte thấp cao của thanh ghi 16 bit.



*Chức năng chính của từng thanh ghi:

- EAX,AX,AL:Được sử dụng trong các lệnh số học và logic,dịch chuyển dữ liệu.
- EBX,BX,BL:Là các thanh ghi lưu địa chỉ.
- ECX,EX,CL:Sử dụng như 1 biến đếm trong các vòng lặp.
- EDX,DX,DL:Ghi dữ liệu cùng với EAX để tham gia vào các thao tác vào ra.

+Nhóm 2: gồm các thanh ghi 32 bit ,16 bit

+32bit:ESI,EDI,EBP,ESP,EIP

+16 bit:SI,DI,BP,SP,IP

*Chức năng

+ESI,SI: Thanh ghi chỉ số nguồn trong khi làm việc với mảng

+EDI,DI: Thanh ghi chỉ số đích trong khi làm việc với mảng

+EBP,BP: Thanh ghi truy xuất dữ liệu trong stack

+ESP,SP: Luôn trỏ đến stack

+EIP,IP: Thanh ghi control lệnh.

+Nhóm 3: Gồm các thanh ghi địa chỉ cơ sở

*CS(code segment) : Lưu địa chỉ phân đoạn mã lệnh.

*DS(Data segment): Lưu địa chỉ phân đoạn dữ liệu.

*ES(Extra Data segment): lưu địa chỉ phân đoạn dữ liệu bổ sung.

*SS(Stack segment): Lưu địa chỉ phân đoạn stack.

Các địa chỉ này kết hợp với offset để truy cập đến ô nhớ.

+Nhóm 4: các thanh ghi cờ(flags) 1bit

*CF(carry flag): Bật khi phép tính có bit nhớ.

*ZF(Zero flag): Bật khi kết quả phép tính =0.

*SF(Sign flag): Bật khi có bit dấu

*OF(overflow flag): Bật khi tràn số.

*PF(parity flag): Bật khi kết quả phép tính có chẵn bit 1.

*AF(auxiliary flag): bật khi phép tính vừa thực hiện có bit nhớ phụ.

*IF(interrupt flag): Bật cờ này để cho có phép ngắt xảy ra.

*DF(direction flag): cờ này bật để chọn chế độ giảm biến tự động.

II. Một số câu lệnh dùng trong chương trình assembly

- %include "win32n.inc" :include file để định nghĩa hằng và struct.
- Extern puts:include hàm puts.
- Import puts msvcrt.dll : hàm puts có trong thư viện msvcrt.dll.
- In_message db "enter a number": khai báo 1 biến xâu kí tự trong segment data.
- Push dword format_in: đẩy biến format_in vào stack. Thanh ghi ESP bị giảm đi 4
- Call[printf] :gọi hàm printf.
- Add esp,4 : tăng giá trị của thanh ghi esp thêm 4. Đẩy các tham số ra khỏi stack.
- Sub edi,esi :edi=edi-esi.
- Mov eax,[N] : đẩy giá trị lưu bởi biến N vào thanh ghi eax.
- Xor edi,edi : Thanh ghi edi=0.
- Div ebx : lấy giá trị của thanh ghi eax chia cho ebx. Thương lưu ở thanh ghi eax, dư lưu ở thanh edx.
- Mov [edi],dl:Di chuyển 1 byte ở thanh ghi dl vào ô nhớ trỏ bởi thanh edi.
- Inc esi :tăng giá trị của thanh ghi esi lên 1,esi=esi+1.
- Cmp eax,0 :so sánh giá trị của eax với 0.
- Je lap2 :nếu toán hạng nguồn = toán hạng đích trong lệnh cmp eax,0 trước đó thì nhảy tới nhãn lap2.
- Loop lap1 :nhảy tới nhãn lap1 nếu giá trị trong thanh ghi cx khác 0.

- Jmp ..start : nhảy không điều kiện tới nhãn start.

III.Chương trình convert Decimal to binary and octan,hexa.

+Mã nguồn C:

```
#include<stdio.h>
```

```
#include<stdlib.h>
```

```
char t[16]={ '0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};    //Xâu để trỏ đến khi thực hiện  
phép modul.
```

```
void binary(int N)
```

```
{
```

```
int i=0,j;                                //Mảng để lưu các mã 2,8,16//
```

```
char *a=(char*)malloc(32*sizeof(char));
```

```
char *b=(char*)malloc(32*sizeof(char));
```

```
char *c=(char*)malloc(32*sizeof(char));
```

```
int M=N;
```

```
while(M>0)                                //Vòng lặp tạo ra mã nhị phân.//
```

```
{
```

```
    a[i]=t[M%2];
```

```
    M=M/2;
```

```
    i++;
```

```
};
```

```
M=N;
```

```
int dem1=0;
```

```
while(M>0)                                //vòng lặp tạo ra mã 8//
```

```
{
```

```
    b[dem1]=t[M%8];
```

```
M=M/8;

dem1++;

};

M=N;

int dem2=0;

while(M>0)                //Vòng lặp tạo ra mã trong hệ 16//
{

    c[dem2]=t[M%16];

    M=M/16;

    dem2++;

};

printf("nhi phan:");        //in ra he 2//

for(j=i-1;j>=0;j--)

printf("%c",a[j]);

free(a);

printf("\nhe bat phan:");    //in ra he 8//

for(j=dem1-1;j>=0;j--)

printf("%c",b[j]);

free(b);

printf("\nhe 16:");          //in ra he 16//

for(j=dem2-1;j>=0;j--)

printf("%c",c[j]);

free(c);

return;

}
```

```
int main()
{
    int N;

    printf("Nhap vao so tu nhien N=");          //Nhap vào số N//

    scanf("%d",&N);

    binary(N);

    return 0;
}
```

+Mã nguồn assembly.

```
%include "win32n.inc"                ;include file

extern puts
import puts msvcrt.dll
extern printf                        ;khai bao cac ham
dung trong chuong trinh
import printf msvcrt.dll
extern scanf
import scanf msvcrt.dll
extern exit
import exit msvcrt.dll

segment .data use32                  ;section data

in_message db "enter a number:",0    ;nhap vao 1 so he 10
out_message1 db "binary is:",0       ;in ra he 2
out_message2 db "octan is:",0        ;in ra he 8
out_message3 db "hexa is:",0         ;in ra he 16
buf db "0123456789ABCDEF",0         ;day tu 0-F de lay
bit
result_2 db "00000000000000000000000000000000",0 ;xau de luu ma
binary, khoi tao 0x00h :32 so 0
result_8 db "000000000000",0 ;xauluu ma octan,khoi tao 12 so 0
result_16 db "00000000",0           ;xau luu ma hexa,khoi tao 8 so 0
                                         ;N-
N dd 0;interger 32bit de luu gia tri nhap tu keyboard
format_in db "%d",0
        ;format_in la xau dinh danh

segment .code use32
```

```
..start:
push dword in_message
call[printf]           ;in ra man hinh :enter a number
add esp,4              ;xoa bo cac tham so ra khoi stack
push N
push dword format_in
call[scanf]            ;goi ham scanf
add esp,8
mov eax,[N]            ;EAX=N
mov ebx,2              ;ebx=2
mov cx,32              ;lap 32 lan
mov esi,0              ;bien dem
lap1:
xor edi,edi
xor edx,edx            ;edx=0
div ebx               ;eax/ebx thuong luu o eax,du
luu o edx
mov dl,[buf+edx]       ;dl=buf[edx]
mov edi,result_2
add edi,31
sub edi,esi
mov [edi],dl          ;dichuyen byte o dl vao result_2 theo thu tu duoi
len
inc esi
cmp eax,0
je lap2               ;Nếu eax=0 thì kết thúc vòng lặp.
loop lap1             ;tiếp tục lặp nếu eax khác 0.
lap2:
mov eax,[N]           ;EAX=N
    ;;khởi tạo lại các thanh ghi để thực hiện với hệ 8
mov ebx,8             ;ebx=8, số chia
mov cx,12             ;lap 12 lan
mov esi,0             ;bien dem
jmp lap22
lap22:
xor edi,edi           ;;xóa dữ liệu trong thanh ghi
edi,edx.
xor edx,edx
div ebx
mov dl,[buf+edx]      ; nạp 1 byte ở buf[edx] vào
thanh ghi dl.
mov edi,result_8
add edi,11            ;biến đếm trở về cuối chuỗi.
sub edi,esi
mov [edi],dl
inc esi              ;tăng biến đếm
```



```
cmp eax,0                ; nếu eax=0 thì thoát vòng lặp.
je lap3
loop lap22                ;lặp nếu eax khác0.
lap3:
mov eax,[N]               ;EAX=N
mov ebx,16                ;ebx=16 ,so bi chia la 16
mov cx,8                  ;lặp 8 lan
mov esi,0                 ;bien dem
lap33:
xor edi,edi
xor edx,edx
div ebx
mov dl,[buf+edx]
mov edi,result_16
add edi,7
sub edi,esi
mov [edi],dl
inc esi
cmp eax,0
je in_2
loop lap33
in_2:                     ;in xau nhi phan
push out_message1        ;đẩy tham số vào stack.
call[puts]
add esp,4                 ;xóa tham số trong stack.
push result_2
call[puts]
add esp,4
in_8:                     ;In ra hệ 8
push out_message2
call[puts]
add esp,4
push result_8
call[puts]
add esp,4
in_16:                    ;in ra hệ 16
push out_message3        ;
call[puts]                ;đẩy các tham số vào stack và gọi hàm
in xau.
add esp,4
push result_16
call[puts]
add esp,4
jmp ..start               ;tiếp tục nhập 1 N khác để convert.
```

+Giải thích về code Assembly:

*Chương trình giúp chuyển đổi các số dương trong hệ 10 từ 0-0xffffffff 32 bit ra các số trong hệ 2,8,16.

*Đầu tiên khai báo các hàm cần dùng trong chương trình :puts,printf,scanf và các thư viện :msvcrt.dll.

*Sau đó khai báo 1 xâu có giá trị từ A-Z để dùng trong phép chia lấy phần dư để gán giá trị cho các mã 2,8,16.

*Tiếp đến khai báo và khởi tạo các xâu của để lưu mã chuyển đổi trong hệ 2,8,16 khởi tạo =0 cho 32 bit.

*Lưu địa chỉ cơ sở của các biến vào các thanh ghi 32 bit để dùng cho vòng lặp

*Thực hiện 3 vòng lặp tương tự nhau:

Trước mỗi vòng lặp phải gán lại giá trị cho các thanh ghi sẽ dùng để không bị ảnh hưởng bởi các vòng lặp trước.

Dựa vào phần dư của các phép chia cho 2,8,16 mà lấy giá trị tương ứng trong xâu buf nạp vào thanh ghi 8 bit(1 byte) dl.Sau đó di chuyển bằng lệnh mov để chuyển giá trị ở thanh ghi dl vào vị trí của các xau result_2,8,16 theo thứ tự từ cuối lên.

*Kết thúc việc lặp thì đẩy các tham số vào trong stack gọi hàm in ra.

*In xong lại quay trở về vòng lặp.

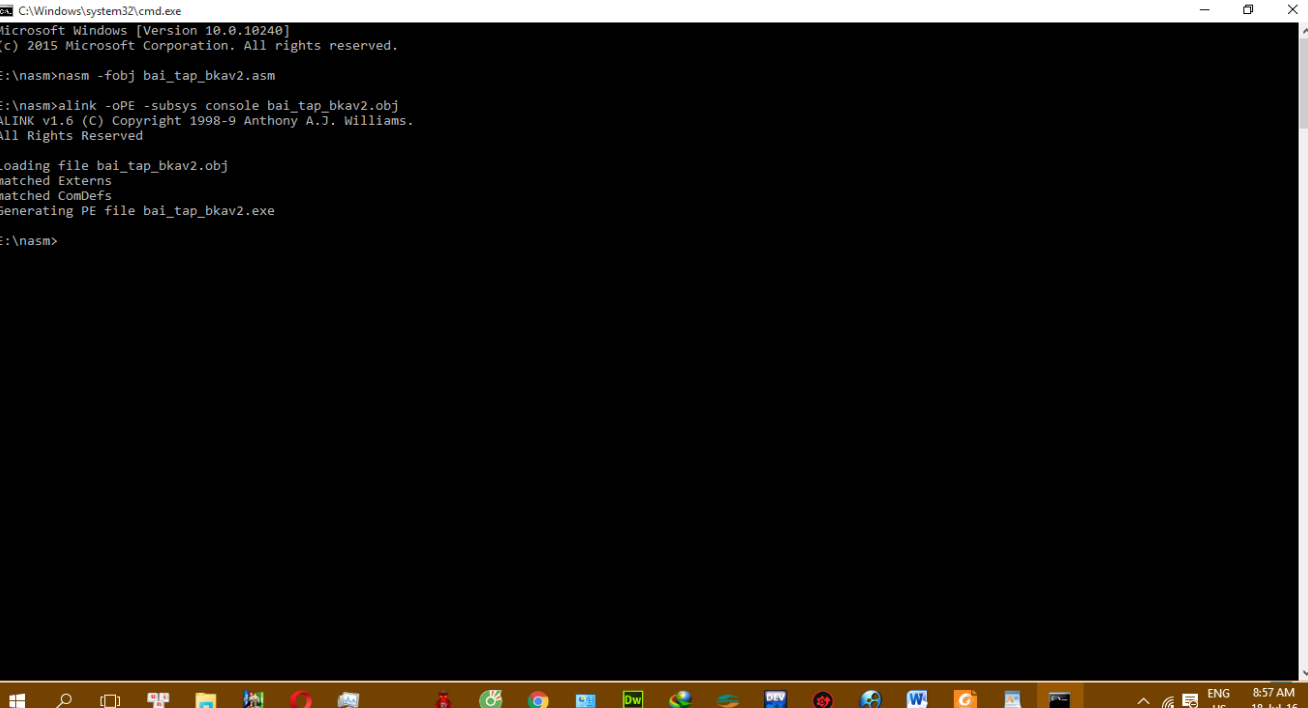
*Sau khi code bằng công cụ notepad lưu dưới với tên bai_tap_bkav2.asm. Lưu trong cùng thư mục với Nasm.

+Trong cmd.exe di chuyển đến thư mục nasm :

Gõ lệnh :nasm -fobj bai_tap_bkav2.asm Để tạo ra file object.

Sau khi tạo file bai_tap_bkav2.obj thành công thì ta thực hiện liên kết các thư viện vào file obj để tạo ra file exe. Sử dụng công cụ alink

Gõ lệnh :alink -oPE -subsys console bai_tap_bkav2.obj



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

E:\nasm>nasm -fobj bai_tap_bkav2.asm

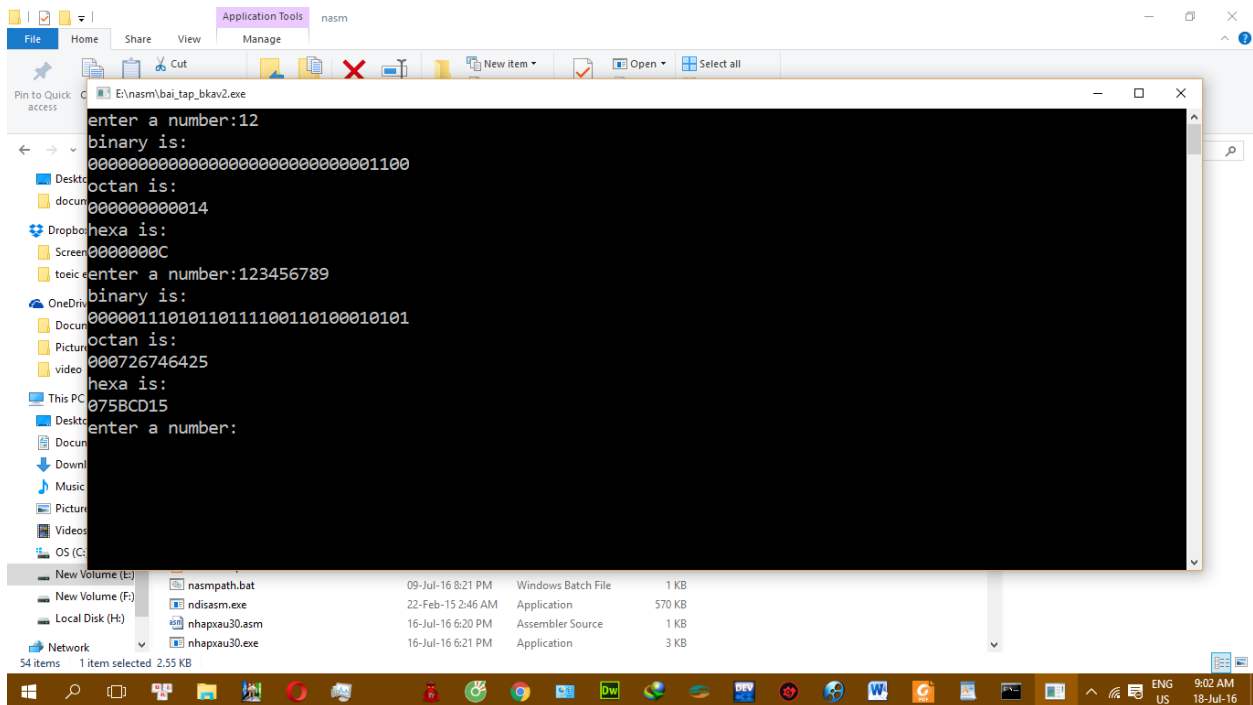
E:\nasm>alink -oPE -subsys console bai_tap_bkav2.obj
ALINK v1.6 (C) Copyright 1998-9 Anthony A.J. Williams.
All Rights Reserved

Loading file bai_tap_bkav2.obj
matched Externs
matched ComDefs
Generating PE file bai_tap_bkav2.exe

E:\nasm>
```

+Sau đó mở file bai_tap_bkav2.exe chạy thử để chuyển đổi số 12 và 123456789 ra hệ 2,8,16

Kết quả sẽ hiện ra như sau:



-----The end-----