Priprava na skusku by LIHO

čo treba vedieť:

najdôležitejšie registre sú: IP = instruction pointer = adresa inštrukcie, ktorá sa má načítať (pointer na ďalšiu) IR = instruction register = inštrukcia, ktorá sa ide vykonať SP = stack pointer = pointer na top zásobníka F = flag = stav procesora po predošlej operácii

segmentové registre alias: Base, Extra, Stack, Data, CODE BAS = bázová adresa segmentu = adresa začiatku segmentu CS = adresa začiatku segmentu, v ktorom je kód DS = adresa začiatku segmentu, v ktorom mám premenné SS = adresa začiatku segmentu, v ktorom mám zásobník ES = nerieš

Logická adresa je určená ako adresa začiatku segmentu (uložená v BAS) a posun voči tejto adrese (relatívna adresa) LA = BAS : RA FA = 16 * BAS + RA (pripomínam, že *16 = 4 krát bitový posun doľava, alebo posunutie o jedno miesto doľava ak to máte vyjadrené ako hexa) logická adresa inštrukcie je teda určená ako LA (inštrukcie) = CS : IP a adresa topu zásobníka ako LA (top zás.) = SS : SP

nepríjemná časť, bohužiaľ treba vedieť: **príznakový register** (F) - - - - O D I T S Z - A - P - C (16 bitov, - = nevyužitý bit) C = carry = nastaví sa na 1, ak nastal prenos z najvyššieho rádu P = parity = nastaví sa na 1, ak výsledok operácie má párny počet jednotiek A = auxiliary = nastaví sa na 1 ak nastal prenos medzi 3. a 4. rádom Z = zero = nastaví sa na 1, ak výsledok operácie je nula S = sign = znamienko, skopíruje sa z najvyššieho rádu výsledku operácie T = trap = na debugovanie, vieš donútiť procesor zastať po každom kroku I = interrupt = hnusný bit, ktorý keď je nastavený na 0 tak procesor nemôže spraviť prerušenie D = direction = smer pri spracovávaní reťazcov (1 je zľava do prava - od vyšších bitov k nižším) O = overflow = nastaví sa na 1 ak došlo k pretečeniu

jednotkový doplnok = otočenie bitov napr. 0101 -> 1010 dvojkový doplnok = otočenie bitov + 1 napr. 0101 -> 1010 + 1 = 1011

priamy kód = normálne vyjadrené číslo s tým, že najvyšší bit určuje znamienko (0 = +, 1 = -) napr. 1111 1111 = -127 (0000 0000 = +0, 1000 0000 = -0) predpätý kód = vždy sa odpočíta 127 nech tam máte čokoľvek napr. 0000 0000 = -127 (predpätý kód, biased, kód s posunutou nulou) doplnkový kód = odpočíta sa 128 keď máte najvyšší bit jednotkový napr. 1000 0001 = -127

BCD kód je desiatkové číslo, že každá cifra je vyjadrená dvojkovo napr. 21 = 0010 0001 -> toto je zhustený formát, že sa do 8 bitov zmestia dve cifry voľný formát = jedna cifra v nižších 4 bitoch, takže sa dá zapísať len <0;9> napr. 3 = 0000 0011 -> toto je voľný, treba pozerať len nižšie 4 bity (tie v pravo :D)

pohyblivá čiarka (IEEE 754 floating point - single precision): reálne číslo vyjadrené v 32 bitoch: z ľava do prava ide SEM: S = sign = znamienko (pripomínam, že 0 = +), E = exponent = 8 bitov, je to v predpätom kóde (to je to, kde sa vždy odpočíta 127) M = mantisa = 23 bitov, v priamom kóde (v normalizovanom tvare, takže 1,...) 0 = 0 00000000 0000000 00000000 00000000 = 00000000 00000000 00000000 00000000 1 = 0 01111111 0000000 00000000 00000000 = 00111111 10000000 00000000 00000000 ak je exponent 0, tak mantisa nie je v tvare 1,... ale 0,... inak je na to tento applet: http://www.h-schmidt.net/FloatApplet/IEEE754.html

problém "pole verzus indiáni": mám číslo v hexa 1A2C a chcem si ho uložiť do 2 prvkového poľa bajtov:

Code: Select all

```
      0  |   1  |      1A  |  2C  | <- takto?      2C  |  1A  |
<- alebo takto?
```

takže to prvé, od najvýznamnejšieho bajtu => Big End First = Big Endian (Motorola) to druhé je ukladanie naopak, od toho nízkeho => Little End First = Little Endian (Intel)

iné:

```
   86tka: 16MB fyzického priestoru    286tka (16bit) : tabuľka
globálnych deskriptorov má maximálnu veľkosť 64 KB, deskriptor je
veľký 8 bajtov,    tabuľka globálnych deskriptorov má 64K/8 = 8K
deskriptorov, lokálnych deskriptorov máme tiež 8K,    čo je
dokopy 16K deskriptorov, každý popisujúci 64KB veľký segment, čo
je 16K*64KB = 1GB virtuálneho priestoru    386tka: 4GB fyzického
priestoru, 64TB virtuálneho (486tka aj Pentium majú tiež 4GB
fyzického)    hľadáte fyzickú adresu, ak DS=1000H a DI=2000H,
spomeňte si na FA = 16*BAS + RA = 16*1000H + 2000H = 12000H    v
selektore -> úroveň privilegovanosti (RPL) = posledná dva bity, T
= 2.bit(3. z prava)    8bitový MUL dáva dolné bity do AL, horné
do AH (celý výsledok do AX proste)    16bitový MUL dáva dolné
bity do AX, horné do DX    8bitový DIV dáva podiel do AL, zvyšok
do AH    16bitový DIV dáva podiel do AX, zvyšok do DX
```

čo Čičák tvrdil, že treba vedieť a vedieť to nebolo treba: alias ako jednoducho počítať veľkosť inštrukcie...

je to **inštrukcia** = +0,5 až +1 bajt kvôli operačnému kódu (OP) je tam označenie pamäti? = +1,5 až +2,5 bajtu označenie pamäti je napr. MOV AX, MOJECISLO; (priama premenná = +1,5B) MOV AX, POLE[SI]; (keď je to prístup k prvku poľa, tak už je to +2,5 B) najväčšie fatality, ktoré viete spraviť je MOV AX, POLE[SI + 12]; je tam označenie registra? = +0,5 až + 1 bajt ... stackuje sa to v takomto poradí: ak máte 2 registre v inštrukcii, tak sa spoja do 1 bajtu, ak tam máte register a označenie pamäti, tak sa register šupne k pamäti -> 0,5 + 1,5/2,5 -> 2B / 3B ak tam máte iba jeden argument a je to register, tak sa to spojí s operačným kódom do 1 bajtu + new! ak tam je iba jeden argument a je to pamäť, tak sa to nespojí s OP -> 0,5 + 1,5 -> 1 + 2 -> 3B je tam konštanta o veľkosti 1B ? (napr. 09H) -> +1B je tam konštanta o veľkosti 2B ? (1234H) -> +2B je tam označenie návestia? = +2 bajty (JMP VYPIS = 3B) všetko! :D máte cca 90% šancu, že sa trafíte

príklady: 1. CLC; OP +0,5 bajtu, nič iné tam neni -> zaokrúhli sa na 1 bajt

2. INC AX; OP +0,5 bajtu, je tam register +0,5 bajtu -> narvú sa do 1 bajtu

3. MOV AX,BX ; OP +0,5 bajtu, prvý register +0,5, druhý +0,5, registre sa spoja do 1B + 0,5 -> zaokrúhli sa na 2B

4. MOV AX, CISLO; OP + 0,5, AX + 0,5, CISLO + 1,5, AX a CISLO sa spoja do 2B + 0,5 -> zaokrúhli sa na 3B

5. MOV AX, POLE[SI + 37]; to isté čo predtým, ale za POLE[SI + 37] sa dá +2,5B -> zaokrúhli sa na 4B

6. XCHG BX, CISLO; +0,5 + 0,5 + 1,5 -> 3B

7. MOV DX, 1234H; +0,5 + 0,5 + 2 -> 3B

8. ADD BX, POLE[SI]; +0,5 + 0,5 + 2,5 -> 4B

9. PUSH AX; +0,5 + 0,5 -> 1B

10. POP CISLO; +0,5 +1,5 -> 2B -> ZLE, bacha to ,5 z označenia pamäte sa do OP bohužiaľ nenarve -> 3B

... atď :D celkom to aj funguje :D