

**Dokumentace KIV/OS­­­­­**

Simulace operačního systému DOS

**prosinec 2016**

Ondřej Dvořák

Tomáš Novák

Radek Vavřík

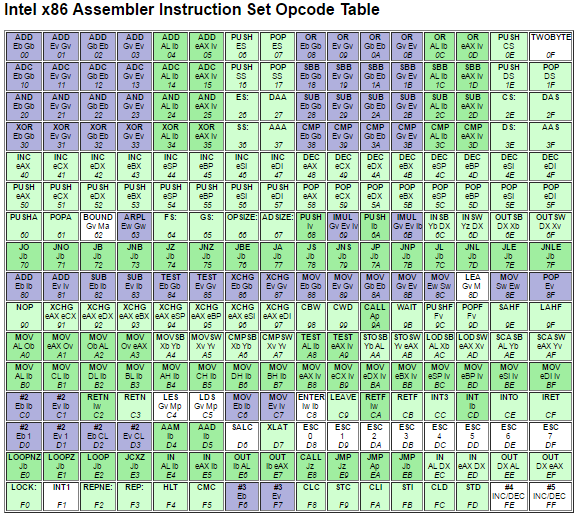
# Zadání

Zvolili jsme si jako tým třetí zadání a to simulaci operačního systému DOS. Zadání jsme si zvolili z důvodu nejlepšího nacvičení a pochopení standartního běhu, jak pracuje operační systém. Chtěli jsme se seznámit jednak s architekturou, pokus se o jednoduchou virtualizaci OS a vyzkoušet si něco jiného než „standartní zadání“.

Součástí zadání byl zdrojový kód a již přeložený soubor COM v 16bit režimu. Pro spuštění bylo potřeba nainstalovat program např. DOSBox. Cílem bylo implementovat minimální emulaci MS-DOSu, která vykoná právě tento program.

# Analýza

Nejdříve jsme se museli začít orientovat v instrukční sadě procesoru Intel x86 a jeho Opcodech – ukázku demonstruje obr:



Na rozdíl od klasického asembleru existuje pro jednu instrukci celá sada opcodů, které se liší parametry. Některé instrukce zapisují přímo do konkrétních registrů, jiné mají zdroj a cíl operace zadány jako parametr. Instrukce se také liší tím, jak kolik bytů načítají jako parametr, s jak velkou částí registru pracují. (Např. zda s celým EAX, nebo jen s AX), zda zapisují na konkrétní paměťové místo přímo a nebo pro výpočet používají další registry. Výpočet takové adresy může být tedy i dán například jako součet DX + ES + hodnota zadaná parametrem. Další kapitolou jsou instrukční prefixy, což jsou prakticky samostatné instrukce, které mění chování následující instrukce. I těchto prefixů může být více. V modelovém příkladu jsme narazilui maximálně na dva zároveň (66 – modifikoval délku registru – tedy že se má pracovat s celým EAX a 26, který říkal, že se nemá použít DS, ale ES pro určení segmentu)

Další rozhodnutí bylo, jakým programovacím jazykem naši semestrální práci budeme vytvářet. Rozhodovali jsme se mezi C# a C. Z důvodu rychlosti, přímého přístupu do paměti, jednodušší správě jsme se rozhodli pro jazyk C.

# Zpracování

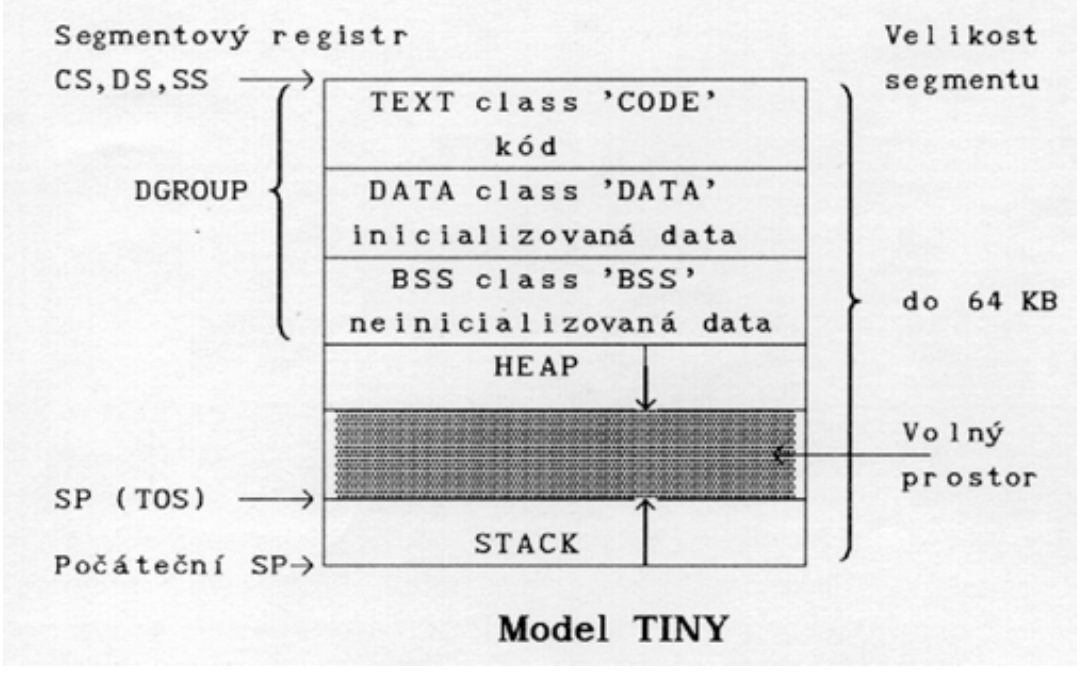
Vývoj programu byl primárně na OS MacOS, debagován v TurboDebageru, testován na Windows 7 64bit, testován ve DOSBoxu.

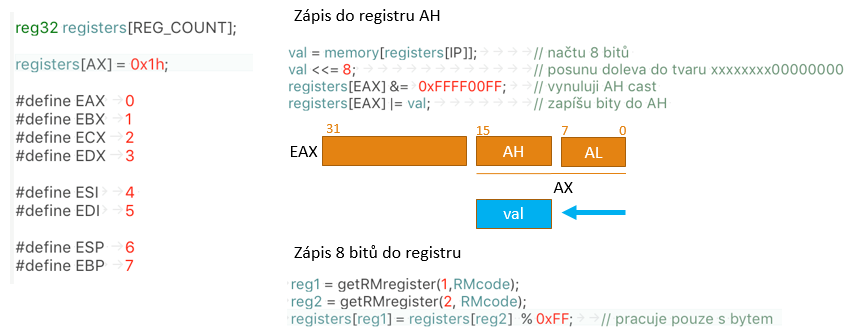
Program prošel několika významnými úpravami. Tyto úpravy pramenily z našich úvah, průběžné prezentace či emailové konzultace se zadavatelem.

Abychom se vyhnuli častým programátorským chybám v alokování paměti, tak jsme nejdříve používali statické pole, nepoužívali pointery. Ve finální verzi jsme potřebný segment alokovali standardně na hromadě a ne v zásobníku.

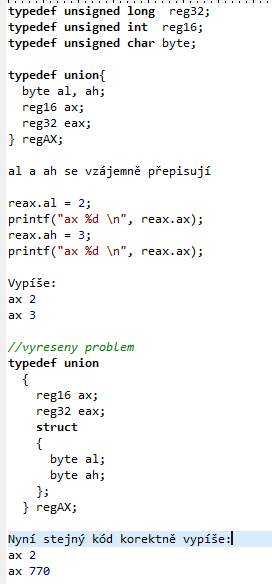
Jelikož byl modelový program v paměťovém režimu TINY, bylo jisté, že si vystačíme s alokací jednoho celého segmentu pro kód i data.

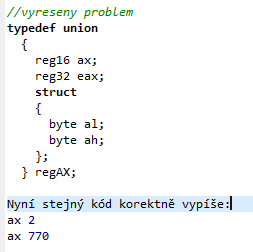
Na počátku jsme si vynulovali registry a nastavili zásobník na konec segmentu. Ten jsme ale nakonec nepoužili.



 Původní logika zápisu do registru byla zvolena takto:

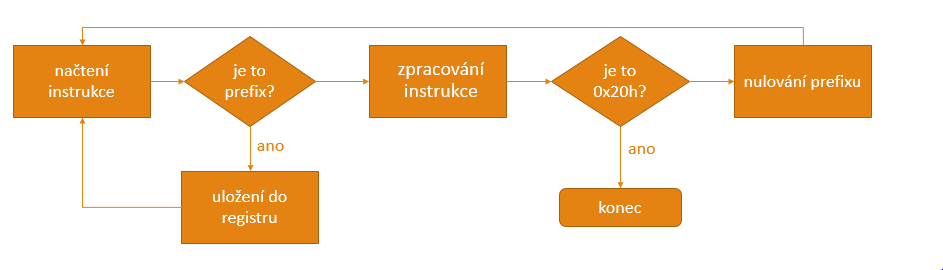
Bylo zvoleno jedno velké 32bitové pole pro všechny registry, do kterého jsme se odkazovali přes indexy definované jako konstanty. Ve finální verzi jsme ale použili union, který je schopen pro jedno paměťové místo použít různé datové typy. Tedy jak 8, 16 tak i 32 bitové číslo. Jediný zádrhel byl, pokud jsme potřebovali do unionu vměstnat dvě proměnné zároveň. Registry AH a AL, protože jsme potřebovali zapisovat i do AH (viz obrázek). Tento problém však bylo možné vyřešit vložením struktury do unionu. Výsledná datová striktura pro jeden registr je tedy následující:



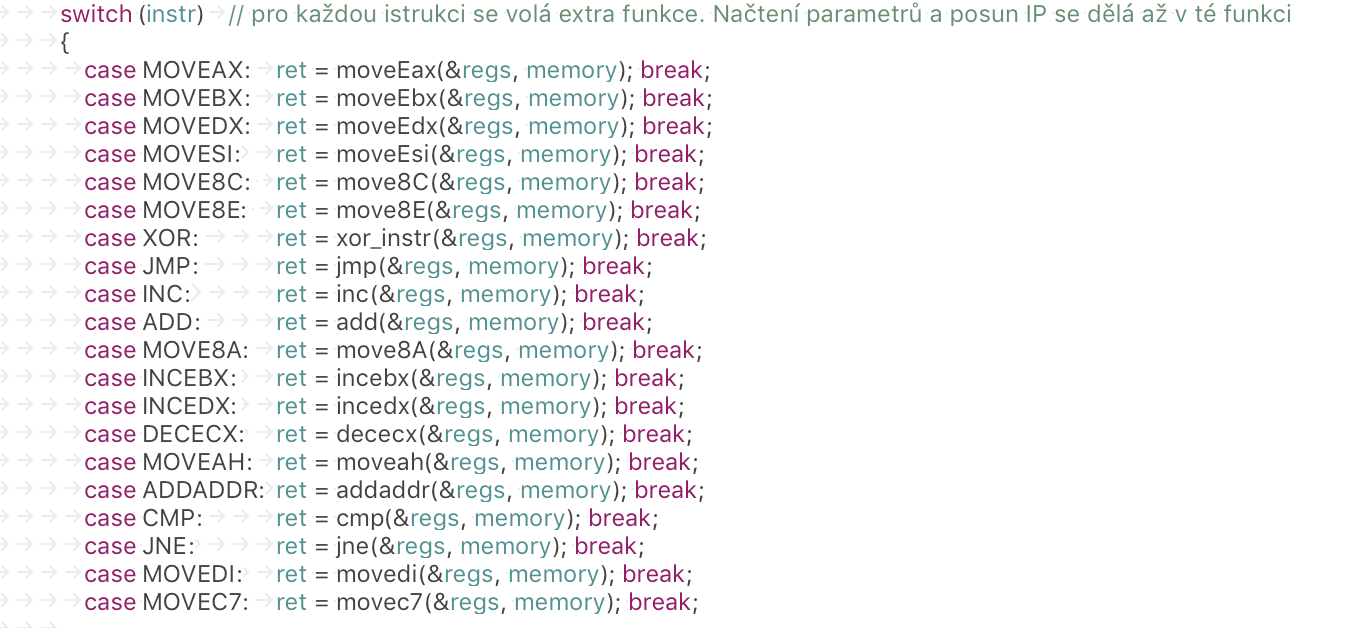


Všechny registry pak byly poskládány do struktury, jejíž pointer se předával mezi jednotlivými funkcemi zpracovávající instrukce.

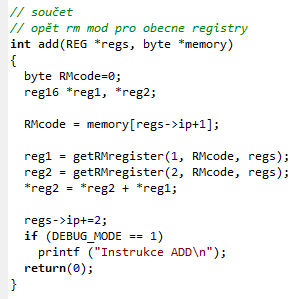
Celkové zpracování programu funguje podle tohoto diagramu:



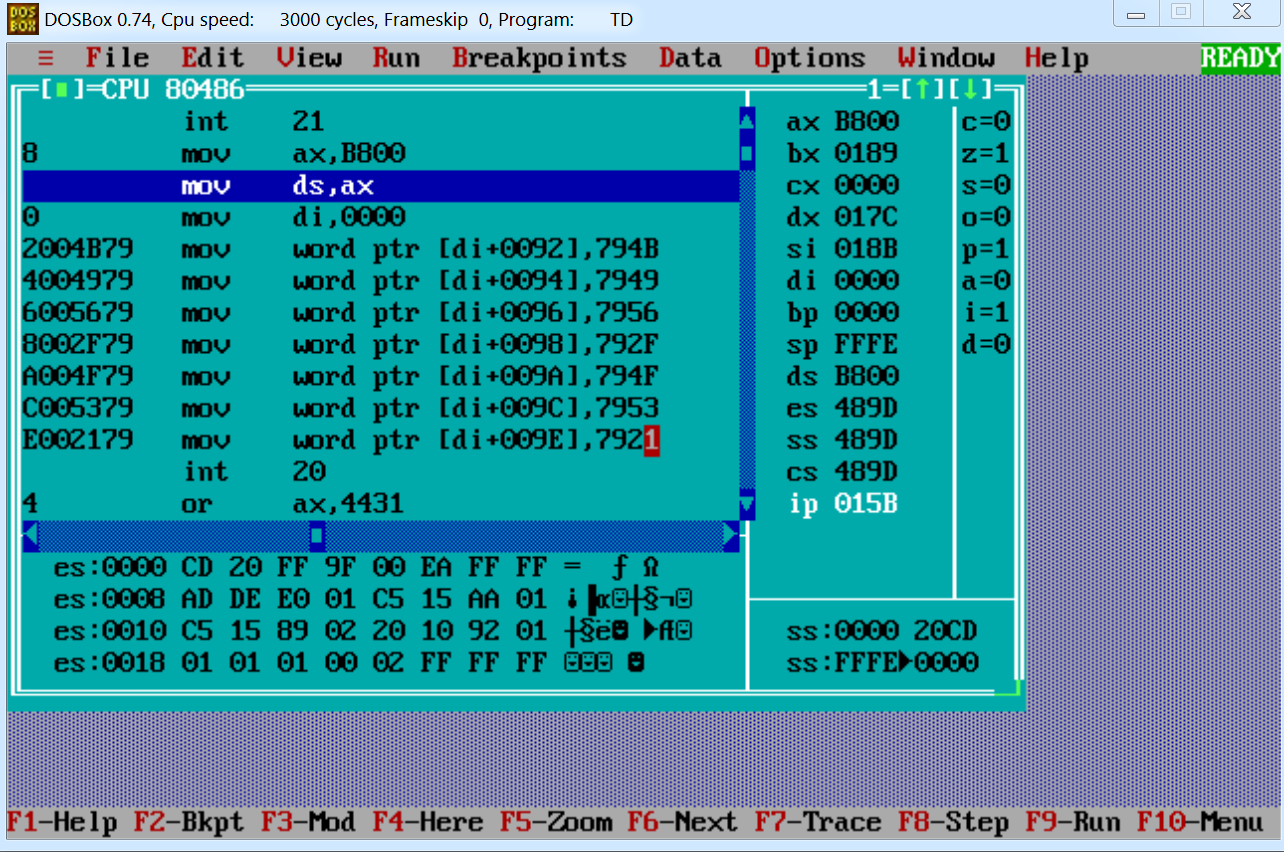
Zpracování jednotlivých instrukcí jsme vyřešili přes velký switch, kde je vidět že jakmile má přijít ke zpracování nějaké instrukce, bude vždy zavolána konkrétní funkce. Parametry instrukce jsou načítány až v konkrétních funkcích, což celý program velmi zpřehledňuje.



Jednotlivé instrukce jsou uloženy v souboru instrukce.c, jsou nazvány dle svého chování a instrukční sady. Pro názornost přikládáme náhled instrukce ADD:



Při vývoji bylo obtížné, že jsme měli poměrně omezenou možnost vidět průběžné výsledky naší práce. Snažili jsme se program debutovat přes zmíněný DOSBox a TurboDebuger viz obrázek níže. V programu také máme tkz. Debug mód, který nám postupně umožnoval „vidět“ co se děje s danými registry a jejich hodnotami.



Program též obsahoval tabulku přerušení, kterou jsme realizovali jako pole ukazatelů na funkce. V modelovém příkladu jsme se setkali s třemi přerušeními.

První z nich (10h) inicializovalo grafický režim v módu VGA 80x25 při 16 barvách, kde na každý znak připadají dva byty. Jeden pro atributy a druhý pro samotný ASCII kód. Po zavolání tohoto přerušení jsme si tedy vynulovali potřebnou paměť začínající na pozici 0x800h.

Další přerušení (20h) volalo službu DOSu pro zápis řetězce ukončeného znakem $ na obrazovku. Jenže náš simulátor nemůže jen tak vypsat něco na obrazovku, protože vzorový řetězec končil odsazením řádku a program v dalších krocích zapisoval na pozice v pravém horním rohu obrazovky. Bylo tedy nutné vytvořit buffer do kterého se zapisovalo. A který se vypsal na obrazovku v okamžiku ukončení programu (21h). Protože jiný podnět mezi posledním zápisem do paměti určené pro obrazovku a koncem programu nebyl.

# Závěr

Všichni jsme si v této semestrální práci zopakovali jazyk C, pochopili jsme a naučili se několik instrukcí pro Intel x86, pochopili základy virtualizace, přerušení, práce s registry a principy fungování operačního systému. Práce byla navíc zajímavá svým nestandartním zadáním, neznámým výstupem/vstupem. Nejvíce se nám líbil skok o -1 krok na parametr předchozí instrukce, který se zpracoval jako instrukce další.

Žel se nám nepodařilo dosáhnout interpretace znakových atributů. Nenalezli jsme jednoduchý způsob jak zapisovat barvy do konzole OSX. (žel conio.h pro mac k dispozici není.) Takže náš výstup zůstává neblikavý a nudně černobílý. Ve všech ostatních parametrech však zcela totožný se zadáním.