FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

Západočeská Univerzita v Plzni

Fakulta Aplikovaných Věd

Operační systémy

Semestrální práce
Simulace operačního systému DOS

David Bohmann - A17N0064P
Jakub Váverka - A17N0095P
Václav Janoch - A17N0070P

26. listopadu 2017

Obsah

1 Zadání

Z nabízených možností jsme si vybrali zadání č. 3: Simulace operačního systému DOS.

Cílem práce bylo seznámení se s instrukční sadou pro procesory navazující na 16bitový procesor Intel 8086, pochopení toho jak probíhá klasický běh programů pod operačním systémem DOS a celkové seznámení s virtualizací.

K zadání byl přiložen jednak spustitelný soubor s koncovkou *.com*, ale i zdrojový kód ze kterého byl vytvořen. Soubor *VB08.COM* je spustitelný pod platformou MS-DOS. Abychom zjistili co přesně dělá, byli jsme nuceni tento soubor spustit v emulátoru DOSBox. Naším úkolem bylo tedy vytvoření takového programu, který by soubor načetl a provedl virtualizaci, abychom získali stejný výsledek, jako DOSBox.

1.1 Přesné znění zadání

- Máte k dispozici malý program pro MS-DOS (com.zip)
  - Včetně zdrojového kódu
  - A včetně kódu přeloženého do COM – tj. pouze sekvence, která se zavede do paměti a CS:IP se nastaví na první zavedený byte
    - Před programem má být 100h bytů dlouhý Program Segment Prefix
  - Program si lze vyzkoušet např. v DOSBox
- Implementujte minimální emulaci MS-DOSu, která vykoná tento program
- Ve své podstatě je úloha o tom, jak se dělá emulace/paravirtualizace

2 Zpracování

Pro programování jsme se rozhodli použít programovací jazyk C, hlavně kvůli jednoduchému přistupování k paměti. Nejprve bylo nutné vytvořit struktury, které by dokázali simulovat strukturu architektury x86.

2.1 Registry

Architektura x86 využívá pro své operace 8 aritmetických registrů, 4 segmentové, instrukční čítač a registr FLAGS. Důležité je si uvědomit, že mohou být operace volány nad různými částmi registrů. Je tedy nutné, abychom zařídili, že po přepsání spodní části registru se změní i hodnota celkového registru. Chtěl jsem tedy vytvořit několik ukazatelů do paměti, které budou ukazovat na ty samá čísla. Výsledek, ale vypadal dost nepřehledně. Naštěstí jsme narazili na takzvané uniony, které dělají v podstatě to samé a výsledný kód je mnohem přehlednější.

2.1.1 Union

Union má v programovacím jazyce C podobný zápis jako struktura. Jeho funkce je však úplně jiná. Místo toho, aby alokoval tolik paměti, kolik potřebuje na všechny své položky, alokuje jen velikost největší. Všechny jeho položky jsou poté umístěny na stejné místo v paměti. Přepsáním jedné, se přepíšou i ostatní. Tímto způsobem se tedy dají velice snadno nasimulovat registry. Na konzultacích jsme zjistili, že jsou takto vytvořené registry součástí zadání číslo 1. My jsme si ostatní zadání tak důkladně nepřečetli a tím pádem jsme si přidělali dost práce s vymýšlením naší struktury pro registry.

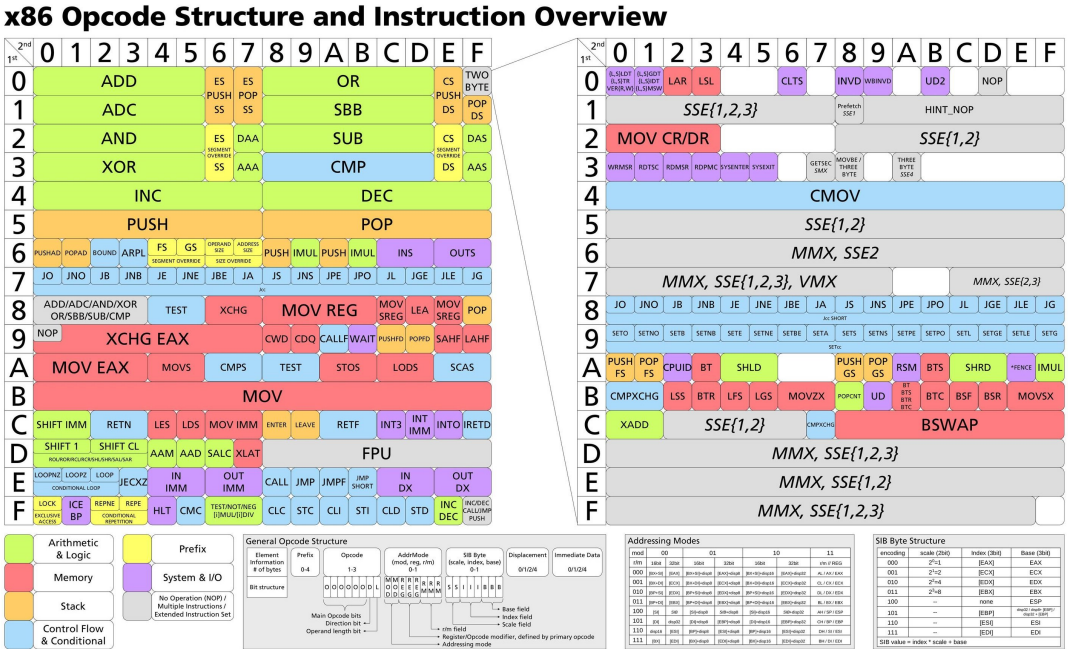
2.2 Paměť

Nejprve jsme plánovali využít jen tolik paměti, kolik bude program skutečně potřebovat a alokovali jsme jen 500 bytů. V průběhu realizace jsme si však všimli, že program zapisuje přímo do video paměti, která je umístěna až na adrese 0xB800, proto jsme navýšili množství alokované paměti na 64 kB.

2.3 Načtení souboru

Po vytvoření struktur pro registry i paměť, jsme se mohli posunout k samotnému načtení do paměti. Z hlavičky souboru se zdrojovým kódem jsme zjistili, že se jedná o tiny režim a že načtený kód je posunutý o 100 bytů od začátku paměti. Načetli jsme tedy soubor *VB08.COM* do paměti a nastavili instrukční čítač *IP* na pozici první instrukce. Dále nás čekala nejtěžší část práce a to bylo pochopení toho, jak fungují opkódy.

2.4 Opkódy



Obrázek 1: Tabulka opkódů

Z tabulky na obrázku 1 lze snadno zjistit, jak se jmenují instrukce příslušící daným opkódům. Následovně bylo nutné nastudovat co přesně daná instrukce dělá a kolik k tomu využívá bytů. Je dobré si povšimnout, že například instrukce *MOV* může být zapsána několika různými opkódy. Nejen *MOV*, ale i ostatní instrukce mohou být dále modifikovány, jednak tímto způsobem, ale i následujícím bytem určujícím adresní mód, bytem SIB a nebo prefixem. Pochopení daných opkódů nebylo jednoduché, naštěstí jsme ale měli k dispozici zdrojový kód a tak jsme měli možnost zkontrolovat, zda se výsledná instrukce rovná té ze zdrojového kódu.

2.5 Instrukce

Rozhodli jsme se každou instrukci realizovat jako samostatnou funkci, která jako parametry dostane celkovou paměť a strukturu se všemi registry. Předtím, než jsme začali programovat samotnou logiku každé instrukce, jsme se je rozhodli pouze všechny načíst, abychom se ujistili, že odpovídají instrukcím ze souboru se zdrojovým kódem. Přišli jsme na to, že bude zapotřebí naprogramovat 22 instrukcí. Jako první instrukci jsme naprogramovali přerušení 20, který ukončí celý program. Díky tomu jsme mohli hned ze začátku udělat smyčku, která je ukončena až touto instrukcí. Uvnitř této smyčky byl zprvu vytvořený *switch*, který rozhodoval, kterou funkci má spustit, podle přečteného opkódu. Po konzultacích jsme tento *switch*, nakonec předělali na *tabulku ukazatelů na funkce*. Kód je tedy tím pádem mnohem čitelnější a jednodušší na pochopení. Volání jednotlivých funkcí vypadá takto

```
functions[(int) memory[dos_registers->ip]](memory, dos_registers);
```

Tabulka 1: Seznam implementovaných instrukcí

Instrukce	Opkód
add_R8_to_RM8	0x00
add	0x03
xor	0x33
increment_EDX	0x42
increment_EBX	0x43
decrement_ECX	0x49
jump_not_equal	0x75
jump_not_parity	0x7B
compare	0x83
move_to_R8	0x8A
move_from_segment	0x8C
move_to_segment	0x8E
move_AH	0xB4
move_AX	0xB8
move_DX	0xBA
move_BX	0xBB
move_SI	0xBE
move_DI	0xBF
move_IMM16_to_RM16	0xC7
interrupt	0xCD
jump	0xEB
increment	0xFF

2.6 Prefixy

Některé z instrukcí využívají prefixy. V našem programu se vykytují prefixy 3. Prefix 26, který značí *Segment override*, 66 značí *Operand override* a 67, který značí *Address override*. Protože náš program načítá instrukce po bytech je nutné si prefix zapamatovat a použít ho při dalším cyklu. Rozhodli jsme se tedy do struktury s registry přidal *flagy*, které ukládají stav prefixů předcházející samotné instrukci.

2.7 Implementace instrukcí

Samotná implementace jednotlivých instrukcí nebyla tak obtížná, jak jsme si zprvu mysleli. Těžké bylo převážně to, že jsme během implementace nevěděli, zda dané instrukce fungují přesně tak jak mají. Většinu instrukcí jsme však naprogramovali správně hned napoprvé a ty které fungovali chybně jsme dokázali snadno opravit s pomocí souboru se zdrojovým kódem.

2.7.1 Skok -1

Implementace skoku do sebe byla překvapivě jednoduchá. Po správné implementaci skoků a přeložení daného opkódu, jsme zjistili, že byt následující skok je opravdu číslo -1. Stačilo tedy přičíst výsledek k instrukčnímu čítači a instrukce proběhla

přesně tak, jak měla.

2.7.2 Přímý zápis do grafického bufferu

Překvapil nás přímý zápis do grafického bufferu. Hlavně kvůli tomu, že jsme měli grafický buffer posunutý na jinou adresu. Po dokončení programu se tedy na obrazovku do pravého horního rohu nic nevypsalo. Naštěstí jsme po prozkoumání paměti řetězec našli a oprava byla jednoduchá.

2.7.3 Přerušení

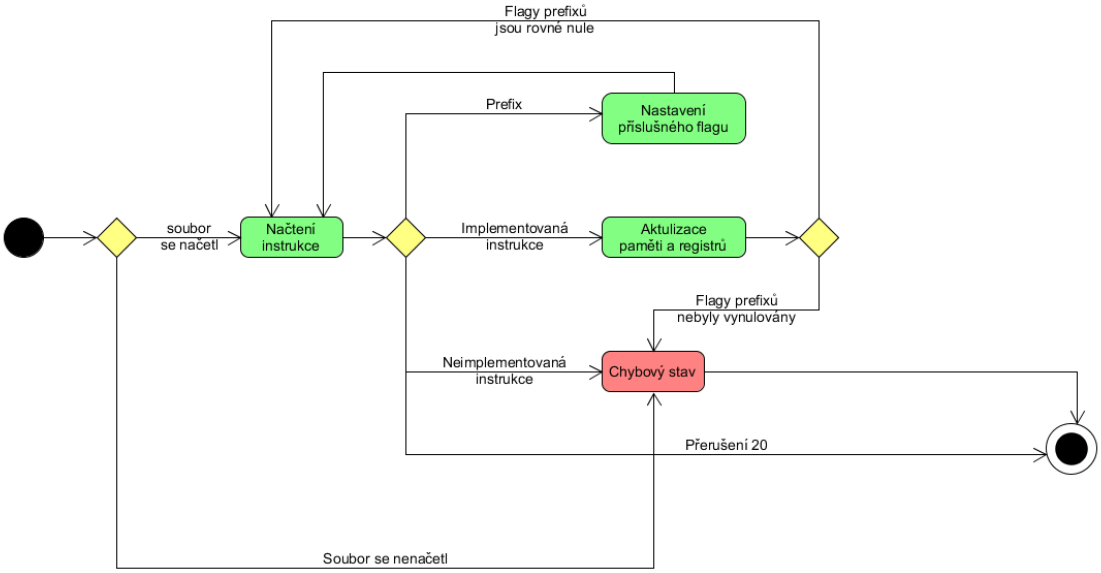
Komplikovaná byla implementace přerušení. Naštěstí se v programu vyskytovali pouze 3. Ukončení programu a zápis do grafického bufferu byl ještě snadný, ale vypsání bufferu na obrazovku nám samotné zabralo několik hodin práce. Příjemné překvapení pro nás bylo, že i v současné konzoli operačního systému Windows 10, je možné vypsat barevně formátovaný výstup, jako tomu bylo u DOSu.

3 Dokumentace

Kód je rozdělen do 5 souborů. Hlavní soubor *main.c*. Tento soubor inicializuje registry a paměť, dále načte do paměti soubor *VB08.COM* a začne provádět jednotlivé instrukce. Poté co přečte instrukci, která odpovídá přerušení 20, ukončí svůj běh a uvolní alokované struktury. Dále se v projektu nacházejí 2 hlavičkové soubory *memory.h* a *registers.h*. V souboru *registers.h* je pomocí uninů definována hlavní struktura registrů. Tuto strukturu pak lze vypsat nebo inicializovat funkcemi ze souboru *registers.c*. Druhý hlavičkový soubor se jmenuje *memory.h* a obsahuje pomocné aliasy instrukcí a deklarace všech instrukčních funkcí. V souboru *memory.c* je pak implementace jednotlivých instrukcí. Samotný běh programu probíhá podle diagramu na obrázku 2.

Tabulka 2: Soubory v projektu

main.c
memory.c
memory.h
registers.c
registers.h



Obrázek 2: Diagram běhu programu

4 Závěr

Práce splňuje zadání v plném rozsahu. Obtížné bylo nastudování a pochopení toho co má vlastně být součástí práce, ale samotná implementace byla poměrně jednoduchá. Při této práci jsme se blízce seznámili s operačním systémem DOS, instrukční sadou x86 a tím jak probíhá emulace.