VEGOVA
ELEKTROTEHNIŠKO-RAČUNALNIŠKA
STROKOVNA ŠOLA IN GIMNAZIJA
LJUBLJANA

Ali so današnji osebni računalniki še vedno kompatibilni z originalnim IBM PC?

Preizkus z lastnim 16-bitnim operacijskim sistemom

Raziskovalna naloga

Računalništvo in informatika

Avtor: Amadej Arh, 3. Letnik SŠ

Mentor: Aleš Volčini, prof.

Ljubljana, marec 2024

Kazalo vsebine

Kazalo slik

Povzetek

V tej raziskovalni nalogi raziskujem kompatibilnost sodobnih računalnikov s prvim osebnim računalnikom in opišem zgradbo in delovanje slednjega. Za ta namen sem v zbirniku spisal 16-bitni operacijski sistem, ki je podoben enoopravilnim operacijskim sistemom 80-ih let prejšnjega stoletja. V nalogi obravnavam znanje potrebno za izdelavo takšnega operacijskega sistema.

Ključne besede: osebni računalnik, 16-bitni operacijski sistem, zbirnik, BIOS

Abstract

In this research paper, I look into the compatibility of modern computers with the first personal computer and describe the structure and operation of the latter. For this purpose, I have written a 16-bit operating system in assembly language, similar to the single-tasking operating systems of the 1980s. This paper discusses the knowledge required to create such an operating system.

Keywords: personal computer, 16-bit operating system, assembly language, BIOS

Uvod

V tej nalogi obravnavam postopke za razvoj primitivnega operacijskega sistema, ki je zasnovan za IBM-ov prvi osebni računalnik - IBM PC 5150. Za tako star računalniški sistem sem se odločil, ker so bili takratni računalniki in operacijski sistemi enostavnejši, že prej pa sem v zbirniku za 8086 napisal klon igre Tetris, ki se je zagnal neposredno po nalaganju zagonskega sektorja diskete. Igro sem kljub drugačnim pričakovanjem uspel napisati zgolj z dokumentacijo in tehnikami tistega časa in izkazalo se je, da so potomci prvega računalnika še vedno sposobni poganjati programe svojih predhodnikov, da so torej z njimi kompatibilni. Želel sem iti še globje in ta operacijski sistem je zelo dober preizkus, kako dobro proizvajalci strojne opreme še sledijo zahtevam po kompatibilnosti za računalnike prejšnjega stoletja in ostarelo programsko opremo.

Hipoteze

Moje hipoteze pri raziskovalni nalogi so sledeče:

1. Proizvajalci strojne opreme so s podporo kompatibilnosti za ostarele strojne in programske opreme zelo dobro sledili vse do nekaj let nazaj.
2. Na novejših računalnikih, proizvedenih po letu 2020 je podpora za BIOS popolnoma odstranjena.
3. Zaradi zelo hitrega razvoja grafičnih kartic imajo grafične rutine težave z popolnostjo.
4. Še vedno na njih lahko zaganjamo operacijske sisteme iz tistega časa.
5. Tudi navidezni računalniki imajo zelo dobro podporo.

IBM PC 5150

V tem delu bom predstavil in opisal specifikacije ciljnega računalnika, ki poganja moj operacijski sistem. To je IBM PC 5150, ki ga je izdelalo podjetje IBM leta 1981 in je postal zgled in standard za naslednje generacije osebnih računalnikov.

Njegove karakteristike so bile naslednje:

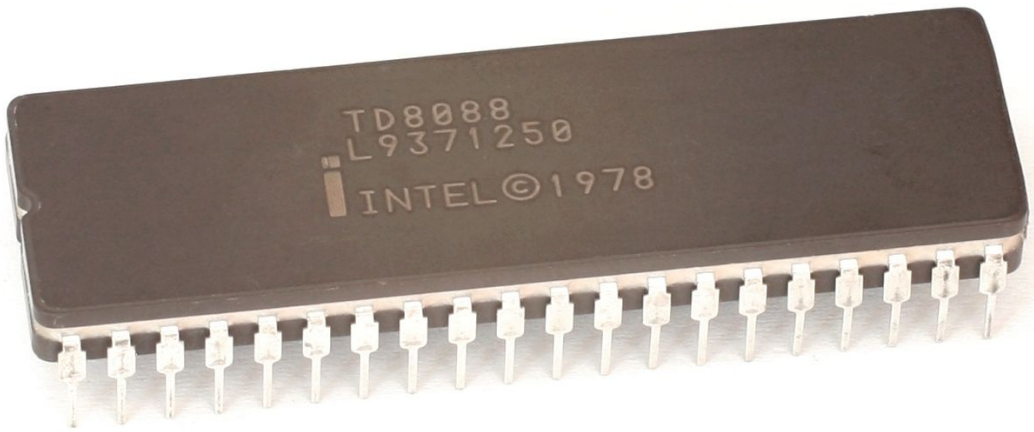
1. Procesor: Intel 8088
2. RAM: 64 KiB
3. 2x 5,25” disketna enota
4. CGA grafična ali monokromatska kartica in kompatibilen monitor
5. tipkovnica

Vredno je omembe, da je to nadgrajena verzija klasičnega PC-ja, saj je ta prišel privzeto s 16 KiB pomnilnika, a ga je bilo možno nadgraditi do 64 KiB in celo do 256 KiB s posebnimi pomnilniškimi karticami. Kupci so lahko izbirali med monokromatskim in barvnim prikazom. Vsaka izmed grafičnih kartic ima namreč svoje prednosti in slabosti, monokromatska ima višjo resolucijo, CGA pa je boljša za igranje iger, saj lahko prikazuje barve.

Slika 1: IBM PC 5150. Vir: Wikimedia Commons

Intel 8088

Je prvi procesor, ki je bil uporabljen na PC platformah, in je cenejša različica Intelovega 8086 procesorja. Razlikujeta se samo v širini podatkovnega vodila, ki je pri procesorju 8088 široko 8 bitov, pri procesorju 8086 pa 16 bitov. Obe varianti procesorja sta 16-bitni, imata 8 splošno-namenskih registrov in se med seboj tudi drugače ne razlikujeta. Registri AX, BX, CX in DX so razdeljeni na 8 bitni višji del in nižji del (npr. AX je sestavljen iz AH in AL). Preostali registri so: kazalec na sklad SP, BP, ki kaže na določeno mesto v skladu in indeksna registra SI ter DI. Posebna registra sta še register z zastavicami in register IP, ki kaže na naslov naslednje inštrukcije, teh se ne da neposredno spreminjati. Zaradi procesorjeve 16-bitne narave bi lahko naslavljali samo do 64 KiB pomnilnika, zato so pri Intelu vpeljali t. i. segmentne registre, s katerimi lahko naslavljamo do 1 MiB pomnilnika v 16-bajtnih korakih in kosih po 64 KiB (16 * 64 KiB je enako 1 MiB). Mednje spadajo register CS (angl. *code segment*), ki dopolnjuje register IP, DS (angl. *data segment*), ki pove na katerem segmentu delamo operacije po pomnilniku, SS (angl. *stack segment*), ki nam pove na katerem segmentu se nahaja sklad, in pomožni segmentni register ES (angl. *extra segment*), ki se lahko uporablja v kombinaciji z splošnonamenskimi registri BX, SI in DI.



Slika 2: Procesor Intel 8088. Vir: Wikipedia

BIOS

BIOS (angl. *basic input/output system*) je program v ROM-u računalnika, odgovoren za zagon le-tega in kot že ime namiguje je zadolžen da programom, ki tečejo na sistemu s pomočjo prekinitev, omogoča nek osnoven način za manipulacijo s strojno opremo (npr. z disketnimi enotami ali tipkovnico).

Prvo opravilo, ki ga BIOS naredi, je POST (angl. *power-on self-test*). V tem delu zagona BIOS stori sledeče:

1. Preizkusi ali registri v CPE delujejo pravilno.
2. Preveri integriteto svoje kode.
3. Inicializira strojno opremo (časovnik, kontroler za prekinitve, vhodno-izhodne naprave, ...).
4. Izmeri velikost pomnilnika in ga testira.
5. Pregleda katere naprave so priključene in na razpolago za zagon.

Nato sledi zaganjanje programa z določene naprave. BIOS to stori tako, da naloži prvi sektor (t. i. zagonski sektor) z izbrane naprave v delovni pomnilnik na naslov 0x7C00 in mu nato preda kontrolo sistema tako da neposredno skoči na ta naslov.

Prekinitve

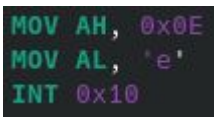
Na splošno so prekinitve uporabljene zato, da se nek časovno kritičen podatek pravočasno obdela. Ko procesor zazna prekinitev dobesedno prekine izvajanje trenutne kode (prvo dokonča izvedbo trenutne instrukcije) in skoči na ustrezno prekinitveno rutino. Običajno se jih deli na:

- Strojne prekinitve.
- Programske prekinitve.

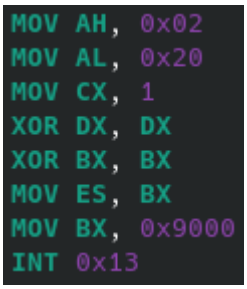
Strojne prekinitve povzroča sprememba stanja strojne opreme na računalniku (npr. časovnik, disketna enota, serijski vmesnik...). Programske prekinitve pa se povzroči preko programa s posebno instrukcijo INT. S temi prekinitvami BIOS omogoča upravljanje z napravami in vmesniki. Pomembnejše prekinitve so:

- Video storitve (prekinitev 0x10).
- Velikost pomnilnika (prekinitev 0x12).
- Storitve disketnih enot (prekinitev 0x13).
- Storitve za tipkovnico (prekinitev 0x16).
- Prekinitve za uro realnega časa (prekinitev 0x17).

Večina BIOS-ovih prekinitev vsebuje različne podfunkcije. Izberemo jih tako, da v register AH naložimo ustrezno vrednost in nato izvedemo prekinitev.



Slika 3: Prekinitev, ki na zaslon izpiše malo tiskano črko e.



Slika 4: Prekinitev, ki prebere prvih 32 sektorjev na prvi disketni enoti in jih shrani na pomnilniški naslov 0x9000.

Na žalost pa si niso iste prekinitve na različnih implementacijah IBM-kompatibilnih BIOS-ov enakovredne. Primer na sliki 3 se ne bo uspešno izvedel na vseh računalnikih, saj količino prebranih sektorjev presega geometrijske meje diska in zmorejo samo nekateri BIOS-i brati preko dimenzij diska, kot jih je poznal prvotni BIOS.

Razporeditev pomnilnika

Prekinitve

V prvih 1280 bajtih pomnilnika se nahajajo prekinitveni vektorji, spremenljivke, ki jih uporablja BIOS. Ta del pomnilnika je pomemben, saj programu omogoča definiranje svojih prekinitev (moramo paziti, da ne povozimo že obstoječih prekinitev, razen če to eksplicitno želimo; sicer pa rutine verižimo).

```
MOV BX, 0x0000
MOV ES, BX
MOV BX, 0x20 * 4

MOV WORD[ES:BX], DOS_INT
MOV WORD[ES:BX + 2], DS
```

Slika 5: Primer nastavljanja prekinitvene rutine »DOS_INT« na prekinitev 0x20.

Pri sliki 5 je velja omeniti, da je med nastavljanjem novih prekinitev pametno začasno onemogočiti prekinitve z instrukcijo CLI, in jih po spremembi ponovno vklopiti z instrukcijo STI. Pri programskih prekinitvah to ni tako pomembno, a pri strojnih že obstaja verjetnost, da nas naprava preseneti s prekinitvijo med nastavljanjem vektorja na novo vrednost.

Video naprave

IBM je za svoj osebni računalnik ob izdaji ponujal monokromatski zaslonski adapter MDA (angl. *monochromatic display adapter*) in barvni grafični adapter CGA (angl. *color graphics adapter*). Obe kartici imata fiksen nabor 256 črk po razširjenem IBM-ovem ASCII standardu, kar pomeni, da ne moremo prikazati raznih šumnikov in drugih slovenskih simbolov. MDA ima resolucijo 720 x 350 in podpira samo tekstovne načine. CGA pa ponuja 2 tekstovna načina in nekaj grafičnih načinov, in sicer:

- 40 črk po 25 vrstic z barvami.
- 80 črk po 25 vrstic z barvami.
- grafični način 320 x 200 pikslov (v izbrani paleti štirih barv).
- grafični način 640 x 200 pikslov (monokromatski način).

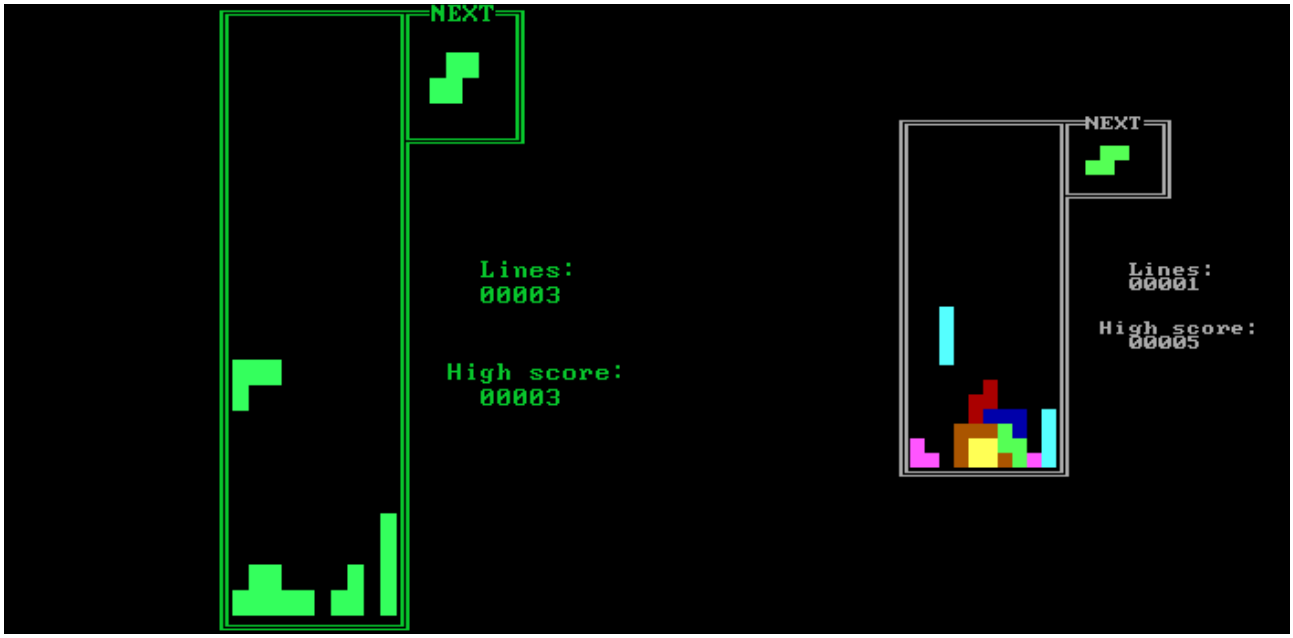
To so vsi uradno dokumentirani video načini, a obstajata tudi 2 nedokumentirana grafična načina (160x200, 160x100), ki imata na voljo 16 barv v zameno za manjšo resolucijo. Video način se nastavi s pomočjo BIOS-ove prekinitve 0x10.

```
MOV AH, 0x00
MOV AL, 0x04
INT 0x10
```

Slika 6: Nastavljanje video načina na 320 x 200 pikslov z štirimi barvami.

Ker imamo za vsak piksel na voljo 4 barve, pomeni da za zapis enega piksla potrebujemo 2 bita. Da bi bili bolj ekonomični z pomnilnikom, so se odločili, da v enem bajtu shranijo 4 piksle, a nam to rahlo oteži pisanje in branje posameznega piksla. Pomnilniški naslov CGA grafik je v tem načinu razdeljen na dva dela (sode vrstice in lihe vrstice), kljub temu da tega ne bi pričakovali. Verjetni razlog je kompatibilnost signalov s televizijskim NTSC standardom.

Na sliki spodaj vidimo mojo implementacijo igre Tetris, ki izrisuje v monokromatskem ali barvnem tekstovnem načinu s pomočjo klicev v BIOS (INT 10h).



Slika 7: Igra Tetris v tekstovnem načinu na levi na monokromatskem zaslonu in na desni z CGA zaslonom.

Datotečni sistem FAT

Datotečni sistem FAT (angl. *file allocation table*) je zelo popularen in hkrati enostaven datotečni sistem. Njegovi začetki segajo v leto 1977, posodobljeni sistemi FAT pa se še danes uporabljajo večinoma za USB ključke, EFI particije in SD kartice. Obstaja več znanih različic tega datotečnega sistema, in sicer:

- FAT12: Uporablja se na disketah.
- FAT16: Uporabljal se je na zgodnjih trdih diskih s kapaciteto do 512 MB.
- FAT32: Uporabljali so ga operacijski sistemi družine Windows 9x, še danes je v uporabi na raznih prenosnih medijih.
- VFAT: Omogoča shranjevanje datotek z daljšimi imeni.
- exFAT: Je posodobljena verzija FAT-a, ki omogoča shranjevanje zelo velikih datotek in se večinoma uporablja na zelo velikih prenosnih medijih.

Struktura FAT-a

Datotečni sistem je razdeljen na 3 oziroma 4 dele:

- Zagonski odsek: Zavzema en sektor, in je vedno na logičnem bloku 0. V njem so shranjeni različni podatki o geometriji diska, velikosti diska, velikosti dodelitvene tabele, velikosti korenske mape... Čisto na začetku vsebuje kratek skok, ki skoči na zagonsko kodo. Sektor se zaključi z magičnima zlogoma 0x55 in 0xAA.
- Dodelitvena (alokacijska) tabela: Velikost je definirana v zagonskem odseku, medij ima ponavadi dve (ali več) takih tabel, da je datotečni sistem bolj trpežen v primeru okvare sektorjev na katerih je shranjena posamezna dodelitvena tabela. Vodi evidenco o zasedenosti gruč (skupin sektorjev) na disku. Te gruče so lahko del datoteke ali mape in ni nujno, da si sledijo druga za drugo kot v datoteki. V dodelitveni tabeli operacijski sistem najde kako si gruče datoteke sledijo v verigi gruč na disku (podobno povezanem seznamu).
- Korenska mapa: Nahaja se tik za dodelitvenimi tabelami. Od FAT32 naprej se nahaja kot mapa v odseku za podatke. Mapa je v resnici datoteka, v kateri je zapisano, katere datoteke (in mape) ji pripadajo, pri kateri gruči se začnejo, koliko so dolge ter ostali atributi.
- Odsek za podatke: Zavzema preostanek diska. Tukaj je v gručah shranjena vsebina datotek.

Branje datotek in map

Če želimo brati in pisati po datotekah in mapah na datotečnem sistemu, moramo prvo imeti način za branje in pisanje po disku. Poznamo pa dva načina za naslavljanje posameznih sektorjev na disku:

- CHS (angl. *cylinder-head-sector*): V tem načinu se posamezen sektor naslavlja tako, da podamo podatke o številki diskovnega valja, glave in sektorja. Ta način je močno vezan na samo geometrijo diska. Včasih je bil bolj pogost, saj je bila geometrija diskov štev no obvladljiva.
- LBA (angl. *logical block addressing*): V tem načinu se posamezen sektor naslavlja tako, da podamo logični blok v katerem se nahaja. Drugače rečeno si v tem načinu sektorji sledijo v zaporednih številkah. Pri IBM PC kompatibilnih sistemih se je na trdih diskih pojavil kasneje.

Seveda je način z logičnim naslavljanjem programsko ugodnejši, ampak na žalost diskete uporabljajo nadležnejši CHS način. Ker so lokacije gruč v dodelitveni tabeli zapisane na LBA način in ker nam bistveno olajša življenje, moramo spisati podprogram, ki dan LBA naslov pretvori v CHS naslov. To se naredi po sledečih formulah:

```
Začasna = (LBA / (število sektorjev na disketni stezi))
Sektor = (LBA  % (število sektorjev na disketni stezi)) + 1
Diskovni valj = (Začasna / (število glav))
Glava = (Začasna % (število glav))
```

Formule so prirejene direktni uporabi v programu, ker instrukcija DIV vrne količnik in ostanek, zato je uporabljena spremenljivka Začasno. Sektorju prištejemo ena, zato ker se v CHS načinu sektorji začnejo z 1 in ne z 0. Nato moramo prebrati dodelitveno tabelo in korensko mapo z pomočjo podatkov v zagonskem odseku. Za robustnost sistema je pametno upoštevati obstoj večih dodelitvenih tabel in program spisati tako, da v primeru napake pri branju tekoče dodelitvene tabele vzamemo naslednjo tabelo.

Korenska mapa pa se nahaja tik za dodelitvenimi tabelami. Njeno velikost v sektorjih je potrebno izračunati in sicer:

```
Velikost korenske mape = (število vhodov * 32
 + velikost sektorja - 1) / velikost sektorja
```

S prištevanjem predhodnika velikosti sektorja zagotovimo zaokrožitev navzgor in število vpisov v imeniku pomnožimo z 32, saj je vsak vpis v imeniku dolg 32 bajtov. Lokacijo se pa izračuna na sledeči način:

```
Lokacija korenske mape = velikost zagonskega odseka
+ število dodelitvenih tabel * velikost dodelitvenih tabel
```

Ko imamo naloženo dodelitveno tabelo in korensko mapo moramo vhod datoteke najti v mapi. Ti so veliki 32 bajtov in držijo razne relevantne podatke kot so:

- Ime datoteke, ki je dolgo 11 bajtov in zapisano z velikimi tiskanimi črkami. Glavni del je dolg 8 bajtov končnica pa 3 bajte, zato je znan tudi kot 8.3 format. Na primer ime »janez.txt« je shranjeno kot »`JANEZ   TXT`«.
- Atributi datoteke, ki zavzemajo 1 bajt. Ti nam povejo ali je datoteka samo za branje, je skrita, je sistemska ipd.
- Bajt rezerviran za Windows™ NT
- Čas ustvarjanja v desetinkah sekunde od 0 do 200.
- Čas ustvarjanja, ki zavzema dva bajta. Sekunde zavzemajo prvih pet bitov z najnižjo težo in so shranjene kot polovična vrednost, nato minute, ki zavzemajo 6 bitov in ure, ki zavzemajo preostalih 5.
- Datum ustvarjanja, ki zavzema dva bajta. Dan zavzema spodnjih 5 bitov, meseci 4 bite in leto 7 bitov.
- Datum prejšnjega dostopa, ki je shranjen v istem formatu kot datum ustvarjanja.
- Zgornja dva bita prve gruče datoteke oz. mape, ki je v FAT12 in FAT16 vedno nič.
- Čas prejšnje spremembe, ki ima je shranjen v istem formatu kot čas ustvarjanja.

- Datum prejšnje spremembe (če verjamete ali ne, je shranjen v istem formatu kot prejšna dva datuma).
- Spodnja dva bajta prve gruče datoteke oz. mape.
- Velikost datoteke v bajtih zavzema 4 bajte.

Vedeti moramo tudi začetek odseka za podatke in sicer:

```
začetek odseka za podatke = velikost zagonskega odseka + velikost
           dodelitvenih tabel + velikost korenske mape
```

Formula za izračun LBA naslova gruče iz dodelitvene tabele:

```
LBA naslov = začetek odseka za podatke + naslov gruče - 2
```

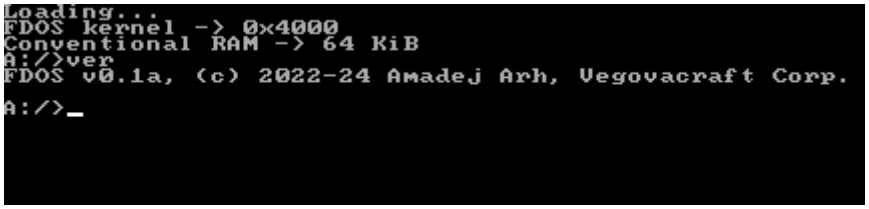
Dva odštejemo, ker se gruče v dodelitveni tabeli začnejo z 2, ne pa z 0. Prvi dve gruči pa sta rezervirani. V dodelitveni tabeli imajo določene vrednosti gruč posebne pomene kot so:

0x000	Označuje prosto gručo na disku.
0x001	Rezerviran za posebne namene, ki jih določi operacijski sistem.
0x002 - 0xFF0	Uporabljen za gruče, ki hranijo podatke datotek oz. map.
0xFF0 - 0xFF6	Nekateri operacijski sistemi jih dojemajo kot dodatne oznake za konec verige.
0xFF6	Rezervirana vrednost.
0xFF7	Označuje pokvarjen sektor v gruči.
0xFF8 - 0xFFF	Označujejo konec verige.

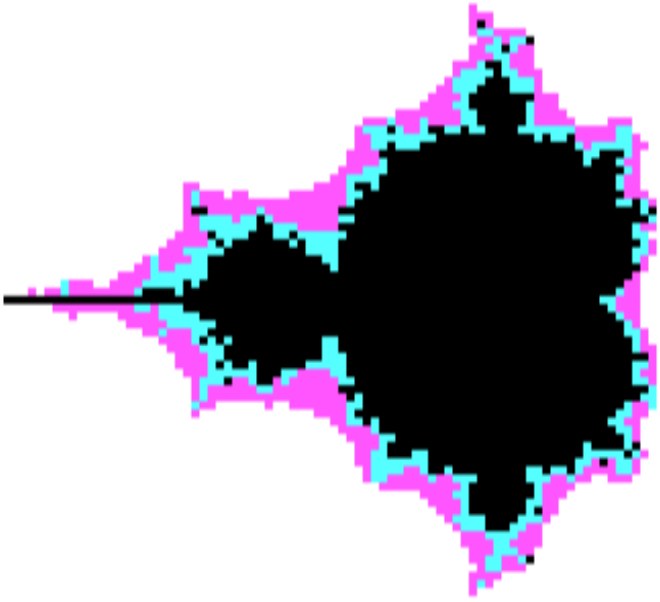
FDOS

FDOS je 16-biten operacijski sistem, ki sem ga napisal za namen raziskovalne naloge in ima podporo za branje in pisanje v datotečnem sistemu FAT12, podpira mape, ponuja nekaj prekinitev za olajšanje dela z CGA in prekinitve za branje in izpisovanje črkovnih nizov ali števil.

Napisan je v zbirniku za Intel 8086/88. Sestavljen je iz zagonskega bloka in datoteke DOS.SYS, podpira datotečni sistem FAT12, osnovno CGA in MDA grafiko, ima vgrajeno lupino z osnovnimi ukazi, lahko zaganja druge operacijske sisteme in poganja programe napisane za FDOS.



Slika 8: Izpis na zaslon ob uspešnem zagonu operacijskega sistema FDOS in izvedba ukaza `ver`.



Slika 9: Izris Mandelbrotove množice s pomočjo FDOS-ove prekinitve, ki demonstrira eno izmed uporab CGA grafik.

Zagon

Na začetku je potrebno nastaviti in inicializirati vse potrebne registre, saj nam BIOS ne zagotavlja, da bodo registri imeli kakršne koli znane vrednosti razen registra DL. Ta hrani število zagonskega medija, ki je nujen za uspešen nadaljnji zagon.

```
START:
        JMP 0x0000:ENFORCE_CS

ENFORCE_CS:
        XOR AX, AX
        MOV DS, AX
        MOV SS, AX
        MOV BP, 0x7C00
        MOV SP, BP
```

Slika 10: Del kode, ki je zadolžen za nastavljanje segmentnih registrov in registra s kazalcem na sklad.

Tudi vrednost programskega segmentnega registra ni pri vseh BIOS-ih enaka. Nekateri BIOS-i skočijo na zagonski odsek z skokom na 0x07C0:0x0000 in ne 0x0000:0x7C00. Kljub temu da oba skoka skočita na isto instrukcijo v naslovnem prostoru velikosti 1 MiB, nam lahko ta prva različica povzroča težave pri relativnih klicih in skokih, ker smo na začetku datoteke zbirniku z direktivo ORG naročili, naj vse naslove obravnava z zamikom 0x7C00 in enako velja za podatke, zato nastavimo podatkovni segmentni register na 0. Register s kazalcem na sklad nastavimo na začetek programskega odseka, saj se kazalec na procesorju 8088 približuje ničli.

Nato s pomočjo prekinitve 13h glavo kontrolerja postavimo na prvo stezo, da se izognemo morebitnim napakam med branjem. V nadaljevanju se lotimo branja dodelitvenih tabel ter korenske mape s pomočjo podprograma, ki lahko bere sektorje na disku z uporabo LBA naslovov. Če nam to uspe, začnemo z branjem jedra FDOS-a, ki je vedno prvi vhod v korenski mapi, zato da nam datoteke ni potrebno iskati. Ko beremo datoteko moramo biti pozorni na zapis gruč v dodelitveni tabeli. Namreč v datotečnem sistemu FAT12 so gruče oštevilčene z 12 bitnimi števili in so v dodelitveni tabeli zapisane zaporedno (v enem bajtu je lahko zapisan del prve gruče in del druge).

Da dobimo vrednost naslednje gruče moramo izračunati njeno lokacijo:

```
Lokacija naslednje gruče = začetek dodelitvene tabele + vrednost
        trenutne gruče + (vrednost trenutne gruče >> 1)
```

Simbol »>>« pomeni bitni pomik v desno in je v tem primeru ekvivalenten deljenju z dva, saj je formula prilagojena za uporabo v programu. Preveriti moramo tudi ali je vrednost trenutne gruče liha ali soda. Če je liha moramo vrednost naslednje zamakniti za 4 bite v desno, za sodo pa moramo odrezati 4 bite z največjo težo z uporabo bitne operacije IN. Ko končamo z branjem verige, v registru DX podamo jedru kazalec na začetek korenske

mape in nastavimo podatkovni segment na vrednost, ki ga uporablja jedro operacijskega sistema in nato skočimo na njegov naslov.

Prekinitev 20h

FDOS svoje storitve ponuja pod prekinitvijo 20h. Te prekinitve nisem izbral iz kateregakoli posebnega razloga, ampak zato ker je bila prva prosta prekinitev in jo uporablja tudi IBM-ov PC-DOS. Storitve izbiramo tako, da v register AH podamo vrednost, ki pripada izbrani storitvi. V času pisanja tega dokumenta obstajajo sledeče storitve:

00h Konča izvajanje programa in se vrne nazaj v lupino.

01h Izpiše niz črk. V registru SI je podan kazalec na začetek niza in v CX podana dolžina.

02h Prebere niz črk. V registru SI je podan kazalec na začetek niza in v CX podana dolžina. Branje se konča, ko uporabnik pritisne tipko Enter.

03h Izpiše nepredznačeno število podano v registru DX v desetiškem sistemu.

10h Prebere podatke iz datoteke shranjene na disku. V paru registrov ES:BX podamo kazalec na mesto, kjer bodo naloženi podatki, v SI kazalec na ime datoteke, v CX število bajtov, ki jih želimo prebrati in v paru DI/DX odmik v bajtih, na katerem začnemo brati, kjer DX predstavlja nižji del in DI višji. Po koncu branja je v registru AL podan statusna koda branja in v CX število bajtov, ki smo jih dejansko prebrali.

11h Zapiše podatke v datoteko shranjeno na disku iz mesta v pomnilniku. Parametre podamo na isti način in status pisanja je podan v istem formatu kot pri storitvi za branje podatkov.

12h Spremenimo našo pozicijo v datotečnem sistemu. V registru SI podamo kazalec, ki kaže na pot, po kateri se želimo premakniti.

13h Izbrišemo datoteko oz. mapo v datotečnem sistemu. Če želimo izbrisati mapo mora biti prazna. Kot parameter podamo kazalec na lokacijo datoteke v register SI.

14h Ustvarimo datoteko, kjer je v registru SI podan kazalec, ki kaže na lokacijo in ime datoteke.

15h Enaka kot prekinitev 14h, ampak za mape.

16h Kopiranje datoteke v drugo mapo. V času pisanja te naloge lahko kopiramo samo eno datoteko naenkrat in ni možnega rekurzivnega kopiranja ali kopiranja map. V registru SI je podan kazalec, ki kaže lokacijo izvorne datoteke, v registru DI pa lokacija ciljne datoteke.

20h Poda vsebino trenutno izvršenega ukaza na naslov podan v registru DI.

21h Storitev, ki izpiše sporočilo o napaki glede na vrednost v registru DL.

30h Nastavi na video način, ki ga FDOS uporablja v lupini.

31h Nastavi trenutni video način na CGA grafični način z 320 vrsticami in 200 stolpci z štirimi barvami.

32h Postavi piksel na ekranu na poziciji x, y. V registru BX je podana x koordinata, v CX y koordinata in v DL barva.

Opazimo lahko, da med prekinitvami obstajajo luknje zato, da bo v prihodnosti lažje dodajati storitve, ki si delijo podobne naloge.

Lupina

Lupina je ena izmed pomembnejših delov operacijskega sistema, saj uporabniku omogoča interakcijo z jedrom, programi in datotečnim sistemom na disku. V našem primeru nam prikazuje lokacijo v datotečnem sistemu po katerem se lahko premikamo z pomočjo ukazov. Ti nam omogočajo opravila kot so brisanje zaslona in izvrševanje programov. Opazili bomo lahko tudi, da si ukazi delijo skoraj podobne oz. identične vloge že obstoječim prekinitvam in služijo le kot vmesnik uporabniku. Ukazi so sledeči:

- BOOT: Zažene zagonski odsek na določenem diskovnem pomnilniku. Kot argument podamo črko diska.
- CD: Omogoča nam pomikanje po datotečnem sistemu. Deluje na enak način kot pripadajoča prekinitev. Kot argument podamo pot, po kateri se želimo premakniti.
- CLS: Pobriše vsebino ekrana.
- CP: Kopira datoteko iz ene lokacije na disku na drugo.
- DIR: Izpiše vsebino mape. Če ga izvršimo brez argumentov izpiše vsebino trenutne mape.
- DSK: Aktivira izvrano disketno enoto oz. pogon.
- LSDSK: Izpiše podatke o obstoječih diskih in disketnih enotah.
- MK: Ustvari datoteko.
- MKDIR: Ustvari mapo.
- REBOOT: Ponovno zažene računalnik.
- RM: Odstrani datoteko oz. mapo. Mapa mora biti prazna, da jo lahko odstranimo.
- TEST: Ukaz namenjen testiranju novih funkcij operacijskega sistema. Uporaben za razvijalca. Za uporabnika je brez pomena.
- TYPE: Izpiše vsebino datoteke.
- VER: Izpiše verzijo operacijskega sistema.

Ena izmed najpomembnejših funkcij lupine je izvrševanje programov. Program lahko izvršimo tako, da podamo njegovo ime brez končnice, ampak mora biti program nujno

označen z končnico PRG. Eksplicitno pa lahko ga lahko izvršimo tako, da damo pred njegovo ime klicaj (namesto `!tetris.prg` lahko napišemo samo `tetris`).

Lupina ne loči med malimi in velikimi tiskanimi črkami.

Programi

Programi v našem operacijskem sistemu niso zgolj surova koda, ampak imajo nekaj podatkov v glavi datoteke, ki povejo operacijskemu sistemu, kam v pomnilnik naj jih naloži. V glavi so tako podatki o verziji, ciljni segment, lokacija skladovnega segmenta, kazalec na sklad in podpis. Zaradi enoopravilne narave operacijskega sistema se lahko ob danem trenutku izvaja samo en program. Naloži se prvih 64 KiB programa. Program pa lahko po potrebi nalaga še ostalo vsebino datoteke. Programi imajo dostop do vseh sistemskih in BIOS-ovih prekinitev. Da končamo izvajanje programa izvedemo FDOS-ovo storitev, ki ustavi izvajanje programa in vrne izvajanje lupini.

Preizkusni računalniki

Na začetku me je zanimalo na katerih računalnikih bom lahko izvajal programe, napisane za najstarejše modele osebnih računalnikov. Izbral sem si nekaj računalnikov, ki sem jih našel doma in tudi navidezne računalnike v okolju QEMU in 86Box. Slednjega sem vzel za osnovno preverjanje ali program sploh v redu napisan in da deluje.

Na računalnikih sem preverjal kompatibilnost tako, da sem prvo preveril, ali računalnik sploh lahko zažene 16-bitni operacijski sistem. Ob uspešnem zagonu sem nato preizkusil delovanje datotečnega sistema tako, da sem naredil kopijo 64 KiB velike datoteke, nato pa še podporo za staro CGA grafiko. Pod posameznim računalnikom ne bom navajal konkretne konfiguracije, ampak le procesor, matično ploščo in grafično kartico. Ker sodobni računalniki nimajo več disketnih enot, sem uporabil disketnik, ki uporablja USB vhod.

Slika 11: Sandberg USB disketna enota

Sama disketna enota ima nekaj težav s kompatibilnostjo ob odstranitvi diskete, saj samo ustavi branje in čaka, da že spet vstavimo nazaj disketo, nato pa bere dalje. Če bi delovala pravilno, bi ob odstranitvi javila napako in ob vstavljanju nove diskete, sporočila operacijskemu sistemu, da se je disketa zamenjala.

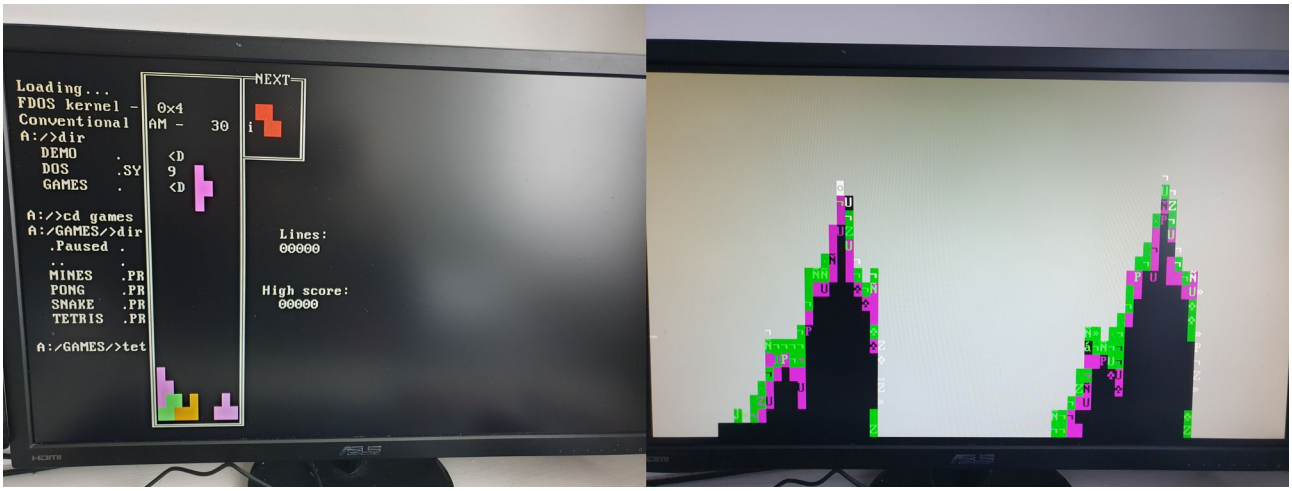
Računalniki, ki sem jih preverjal so sledeči:

Moj osebni računalnik

Slika 12: Računalnik z matično ploščo ASUS Maximus Ranger VII izdelana leta 2014, s procesorjem Intel i7-4790 in grafično kartico Nvidia RTX 2060 revizija a1.

Leto izdelave	2014
Matična plošča	ASUS Maximus Ranger VII
Procesor	Intel i7-4790
Grafična kartica	Nvidia RTX 2060

Računalnik je zagnal operacijski sistem. Operacije na datotečnem sistemu delujejo brez problemov. Na žalost pa grafična kartica slabo podpira stare grafične načine. Ob zagonu programa Tetris, se prejšnja vsebina ekrana ni pobrisala in sam tekstovni način ni bil pravilno prikazan. Pri izrisu mandelbrotove množice pa grafičen način sploh ni bil pravilno prikazan. Operacijski sistem je lahko zaznal vse vgrajene diske, vključno z NVMe diskom.



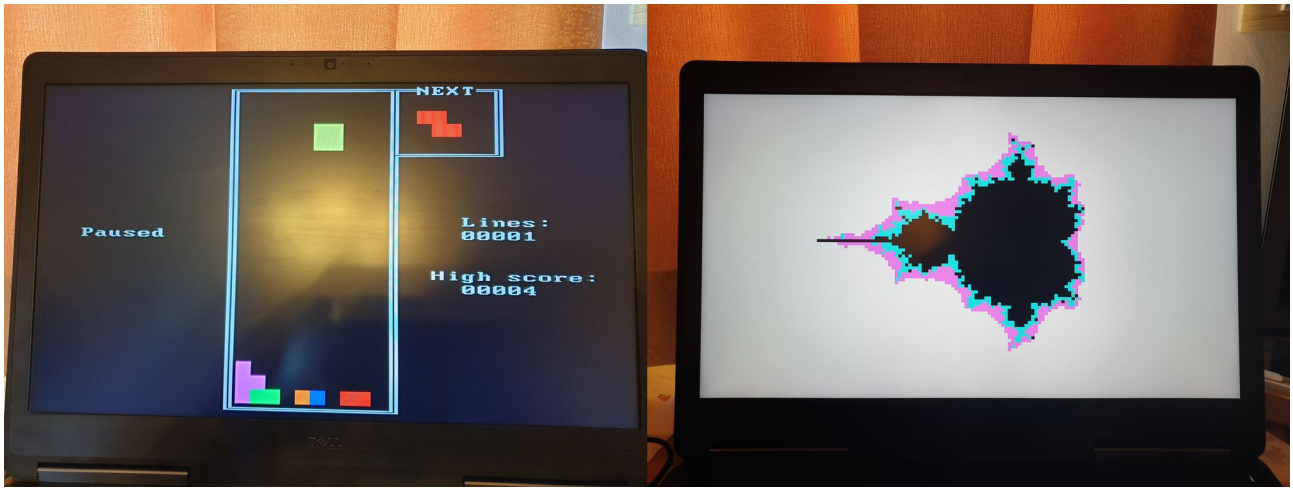
Slika 13: Na levi strani je prikaz Tetrisa z nepočiščeno prejšnjo vsebino, na desni pa nepravilna podpora grafičnega načina.

Prenosnik DELL Precision 7710

Slika 14: Prenosnik DELL Precision 7710 izdan leta 2019, ki ga poganja procesor Intel i7-6820HQ z integriranimi Intel HD 530 grafikami.

Leto izdaje	2019
Ime računalnika	DELL Precision 7710
Procesor	Intel i7-6820HQ
Grafike	Integrirane Intel HD 530

Računalnik ni imel težav z zagonom, razen da na začetku spusti eno vrsto. Vse operacije po datotečnem sistemu in kopiranje testne datoteke je delovalo kot pričakovano. Integrirana grafična kartica dobro podpira CGA grafične načine. Operacijski sistem je zagnal vse pogone na računalniku.



Slika 15: DELL prenosnik, ki poganja Tetris in izris mandelbrotove množice.

IBM Aptiva

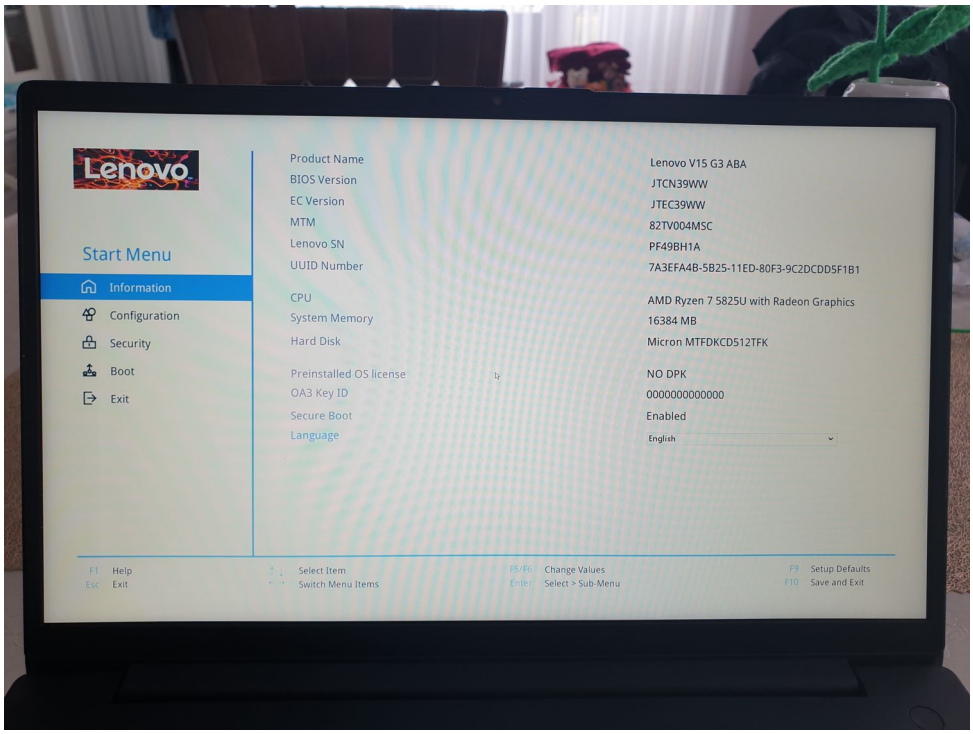
Slika 16: Računalnik IBM Aptiva, proizveden leta 1999. Poganja ga procesor AMD K6-2 500MHz in integrirana AGP grafika SiS 530.

Leto izdelave	1999
Ime računalnika	IBM Aptiva
Procesor	AMD K6-2 500MHz
Grafika	Integrirana AGP SiS 530

Pri tem računalniku nisem pričakoval nobenih težav z kompatibilnostjo. Ker ima vgrajeno podporo za disketne enote tukaj ni prišlo do težav pri javljanju napak ob odstranitvi diskete. Pisanje in branje datotek po datotečnem sistemu deluje kot pričakovano. Grafična načina sta dobro podprta in sistem zazna vse pogone v računalniku.

Slika 17: računalnik IBM Aptiva, ki poganja Tetris in izris mandelbrotove množice.

Prenosnik Lenovo V15 G3

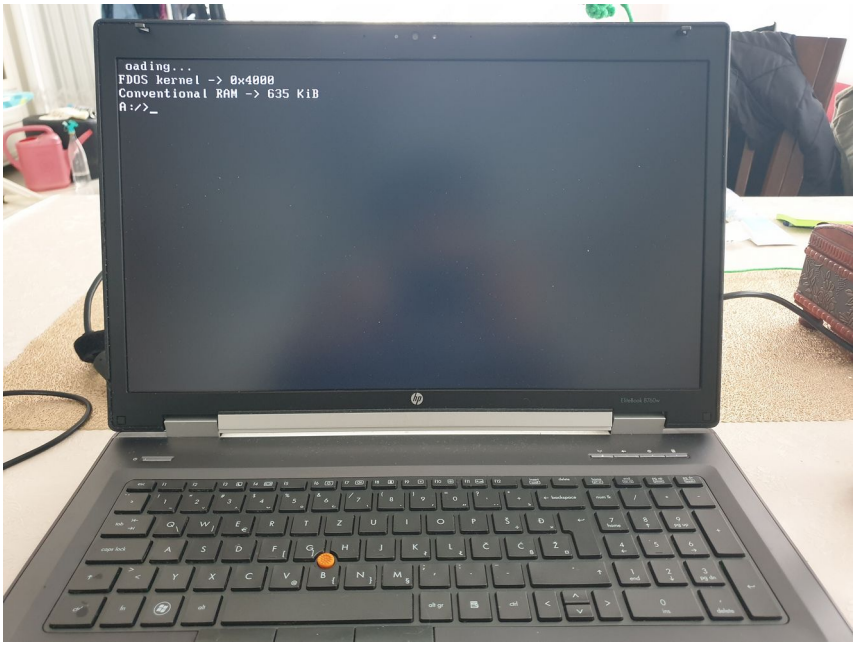


Slika 18: Prenosnik Lenovo V15 G3 izdan leta 2023, ki ga poganja procesor Ryzen 7 5825U z integriranimi grafikami.

Leto izdaje	2023
Ime računalnika	Lenovo V15 G3
Procesor	Ryzen 7 5825U
Grafike	Integrirana grafika

Prenosnik ne podpira zagona z BIOS načina in posledično ne more zagnati operacijskega sistema. Kompatibilnost z ostarelo opremo je popolnoma odstranjena.

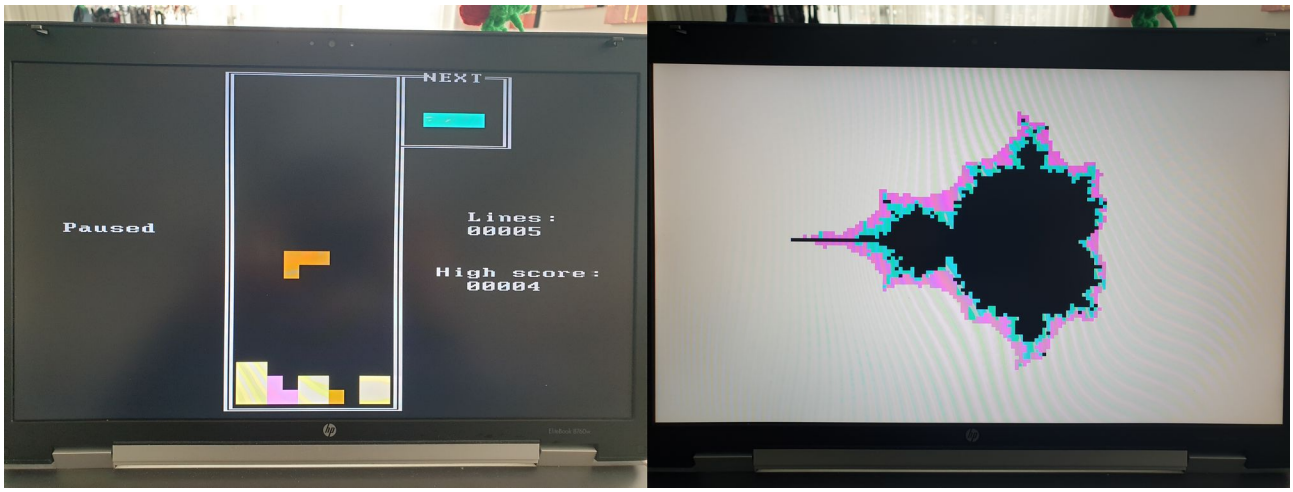
Prenosnik HP Elitebook 8760w



Slika 19: Prenosnik HP Elitebook 8760w izdan leta 2011, ki ga poganja procesor Intel i5-2540M in grafična kartica AMD M5950.

Leto izdaje	2011
Ime računalnika	HP Elitebook 8760w
Procesor	Intel i5-2540M
Grafična kartica	AMD M5950

Računalnik zažene operacijski sistem šele po drugem poskusu. Pri prvem se pojavi napaka pri branju z statusno kodo 0, ki naj bi pomenila, da ni bilo napak. Ob zagonu se prva črka vedno izpiše kot prazna črka. Po datotečnem sistemu lahko normalno pišemo ali beremo, dokler se ne vrnemo iz programa nazaj v operacijski sistem. Takrat je potreben ponoven zagon, saj ne moremo ne brati ne pisati po disketi. Podpora za grafike deluje brez težav. Operacijski sistem zazna vse diske, ampak disketna enota prikaže, da ni vstavljena disketa, kljub temu da je.



Slika 20: Grafike na prenosniku HP, ki pravilno izrišejo Tetris v CGA tekstovnem načinu in mandelbrotovo množico.

Zaključek

Vse računalnike sem ocenil z dvema kriterijema. To sta podpora za branje in pisanje po disketah in podpora za grafiko. Ocenil sem jih z ocenami, ki pomenijo, da računalnik:

0 sploh ne podpira funkcionalnosti.

1 podpira to funkcionalnost, ampak zelo slabo.

2 dobro podpira funkcionalnost, a so še razvidne napake.

3 funkcionalnost podpira brezhibno.

Računalniki	Leto	Podpora za branje in pisanje po disketah	Podpora CGA grafiki	Vsota ocen
Moj osebni računalnik	2014	2	1	3
DELL Precision 7710	2019	2	2	4
IBM Aptiva	1999	3	3	6
Lenovo V15 G3	2023	0	0	0
HP Elitebook 8760w	2011	1	2	3

V zgornji tabeli lahko vidimo, da računalniki do leta 2019 in morda še kakšno leto dlje še vedno nekako ali celo zelo dobro podpirajo funkcionalnosti, ki izvirajo iz originalnega IBM PC iz leta 1981, torej celih 38 let. Proizvajalci so torej ves čas vztrajali pri združljivosti sistemov s strojno opremo, ki že zdavnaj ni več v uporabi. Ne pozabimo, da je ta računalnik ob izidu imel le 16 KiB pomnilnika RAM in 16-bitni procesor. To ob enem pomeni, da sta podjetji Microsoft in Intel vlagali izredne napore v kompatibilnost s starimi sistemi tudi še dolgo po izidu 32-bitnih operacijskih sistemov. Glede na hitrost razvoja računalnikov je to trajalo presunljivo dolgo. Intel je šele pred kratkim začel razmišljati o opustitvi naborov 16 in 32-bitnih instrukcij. Kompletno programsko kodo operacijskega sistema FDOS-a si lahko ogledate preko povezave https://github.com/Ama-dej/FDOS.

Kaj smo se naučili

Večino mojih hipotez sem v tej raziskovalni nalogi potrdil in sicer:

1. Večina računalnikov, ki sem jih preizkusil je še vedno kompatibilnih ali imajo vsaj osnovno podporo s prvim osebnim računalnikom.
2. Hipoteza, da se kompatibilnost začenja opuščati je bila tudi potrjena, saj najnovejši računalnik ni mogel zagnati 16-bitnega operacijskega sistema.
3. Grafične rutine nimajo toliko težav kot sem prvotno mislil, a še vedno niso popolne, še posebej pri računalniku z diskretno grafično kartico.

4. Večina računalnikov lahko še vedno zažene 16-biten operacijski sistem razen enega.
5. Navidezni računalnik QEMU ima še vedno zelo dobro podporo, a sem med izvajanjem naletel na par napak, kar me niti ni presenetilo, saj je QEMU namenjen virtualizaciji modernejših sistemov. Največja težava je pa bila, da QEMU ne more virtualizirati vse strojne opreme prvega osebnega računalnika kot so MDA in CGA grafika. Problem je bil tudi, da ni bilo mogoče nastaviti manj kot 1 MiB pomnilnika, saj bi 64 KiB moralo biti za vsakega dovolj.

Večina jih podpira vsaj osnovne rutine, da lahko operacijski sistem dela, a takoj ko začnemo preizkušati in opazovati podrobnosti ugotovimo, da kompatibilnostna plast ni popolna. Ugotovil sem tudi, da je podpora za branje in pisanje po disketah tudi odvisna od same USB disketne enote.

Če bi se zadeve lotil od začetka, bi spremenil veliko o samem operacijskem sistemu, saj sem se s to raziskovalno nalogo veliko naučil o programiranju. Kot prvo bi ukaze shranjeval v mapi in bi s tem poraba pomnilnika manjša. Nekatere procedure za branje in pisanje v samem jedru niso spisane z podporo za več različnih diskovnih medijev in same prekinitve nimajo podpore za zaporedno branje oz. pisanje.

Kritika realnega načina

Glavna kritika tega načina je da, ko postane operacijski sistem bolj razvit nas BIOS-ove storitve začnejo ovirati, saj samo jemljejo strojne vire. Na primer ne moremo spisati svojega gonilnika za disketne enote brez da bi ovirali BIOS-ove rutine. Želeli bi jih pa zamenjati zato, ker ne omogočajo vseh funkcij, ki jih tista naprava oz. naprave podpirajo. Zavedam se, da je to pogled iz moderne perspektive, ker je takrat bila to primerna rešitev zaradi omejene količine pomnilnika.

Orodja

Pri izdelavi operacijskega sistema sem uporabil razna orodja. Za prevajanje zbirnega jezika sem uporabil prevajalnik NASM (angl. *Netwide Assembler*). Za testiranje sistema sem uporabil razna orodja za virtualizacijo. Sprva sem uporabljal emulator QEMU, ampak sem ugotovil, da je bolj namenjen za učinkovite virtualne stroje in ne emulira vseh podrobnosti in napak sistema, zato sem se odločil za uporabo emulatorja 86Box. Ta ima večji poudarek na natančni emulaciji ciljnega sistema in je bolj namenjen starejšim strojem od prvega osebnega računalnika do računalnikov s procesorjem Pentium 3. Delovanje sistema sem med razvijanjem preverjal na pravi strojni opremi. Kodo in vsebino raziskovalne naloge sem pisal v namiznem okolju Linuxa.

Viri

[1] IBM. **5150 Technical Reference.**
Dostopno na: https://minuszerodegrees.net/manuals/IBM_5150_Technical_Reference_6025005_AUG81.pdf (3. 3. 2024)

[2] IBM. **IBM Color/Graphics Monitor Adapter.**
Dostopno na: https://minuszerodegrees.net/oa/OA%20-%20IBM%20Color%20Graphics%20Monitor%20Adapter%20(CGA).pdf (3. 3. 2024)

[3] IBM. **IBM Monochrome Display and Printer Adapter.**
Dostopno na: https://minuszerodegrees.net/oa/OA%20-%20IBM%20Monochrome%20Display%20and%20Printer%20Adapter.pdf (3. 3. 2024)

[4] Intel. **The 8086 Family User's Manual.**
Dostopno na: https://bitsavers.org/components/intel/8086/9800722-03_The_8086_Family_Users_Manual_Oct79.pdf (3. 3. 2024)

[5] Microsoft. **Microsoft Extensible Initiative FAT32 File System Specification.**
Dostopno na: https://download.microsoft.com/download/1/6/1/161ba512-40e2-4cc9-843a-923143f3456c/fatgen103.doc (3. 3. 2024)

[6] Intel. **Envisioning a Simplified Intel Architecture.**
Dostopno na: https://www.intel.com/content/www/us/en/developer/articles/technical/envisioning-future-simplified-architecture.html (3. 3. 2024)

Priloga

Zagonski nalagalnik (angl. *bootloader*)

```
CPU 8086
[BITS 16]
[ORG 0x7C00]
%INCLUDE "src/locations.h"

BPB:
SHORT_JUMP:
       JMP SHORT START
       NOP
OEM: DB "FDOS    "
BYTES_PER_SECTOR: DW 512
SECTORS_PER_CLUSTER: DB 1
RESERVED_SECTORS: DW 1
NUMBER_OF_FAT: DB 2
ROOT_ENTRIES: DW 224
SECTOR_COUNT: DW 2880
MEDIA_DESCRIPTOR: DB 0xF0
SECTORS_PER_FAT: DW 9
SECTORS_PER_TRACK: DW 18
HEAD_COUNT: DW 2
HIDDEN_SECTORS: DD 0
LARGE_SECTOR_COUNT: DD 0

EBR:
DRIVE_NUMBER: DB 0
RESERVED: DB 0
SIGNATURE: DB 0x29
VOLUME_ID: DD 0xFFDD0055
VOLUME_LABEL: DB "FDOS       "
IDENTIFIER_STRING: DB "FAT12   "

START:
       JMP 0x0000:ENFORCE_CS

ENFORCE_CS:
       XOR AX, AX
       MOV DS, AX
       MOV SS, AX
       MOV BP, 0x7C00
       MOV SP, BP

       MOV BYTE[DRIVE_NUMBER], DL ; Store the drive number from the BIOS.
       INT 0x13 ; Reset drive head.

       ; MOV AH, 0x08
       ; INT 0x13 ; Get diskette info.

       ; MOV AL, CL
       ; XOR AH, AH
       ; AND AL, 0x3F
       ; MOV WORD[SECTORS_PER_TRACK], AX

       ; INC DH
       ; SHR DX, 8
       ; MOV WORD[HEAD_COUNT], DX

       MOV ES, AX

       MOV SI, LOADING_MSG
       CALL PUTS

       MOV DH, BYTE[NUMBER_OF_FAT]
       MOV CX, WORD[SECTORS_PER_FAT]

       MOV AX, WORD[RESERVED_SECTORS]
       MOV BX, FILESYSTEM
       MOV DL, BYTE[DRIVE_NUMBER]

READ_FAT_LOOP: ; In case the first fat table is broken try to load the redundant ones.
       CALL READ_DISK
       JNC .OK

       ADD AX, CX

       DEC DH
       JZ ERROR
```

```
        JMP READ_FAT_LOOP

.OK:
        MOV AL, BYTE[NUMBER_OF_FAT]
        XOR AH, AH
        MUL WORD[SECTORS_PER_FAT]
        PUSH AX

        ADD AX, WORD[RESERVED_SECTORS]

        MOV BX, CX
        ; SHL BX, 9
        SHL BX, 1
        SHL BX, 1
        SHL BX, 1
        SHL BX, 1
        SHL BX, 1
        SHL BX, 1
        SHL BX, 1
        SHL BX, 1
        SHL BX, 1
        ADD BX, FILESYSTEM
        MOV WORD[ROOT_DIR_LOC], BX

        MOV CX, WORD[ROOT_ENTRIES]

        ; SHL CX, 5
        ; ADD CX, 511
        ; SHR CX, 9

        ADD CX, 15
        SHR CX, 1
        SHR CX, 1
        SHR CX, 1
        SHR CX, 1

        PUSH CX

        MOV DL, BYTE[DRIVE_NUMBER]

        CALL READ_DISK
        JC ERROR

        MOV AX, WORD[BX + 26] ; Get the first sector of the DOS.SYS file.
        MOV WORD[EXPLORER_FIRST_SECTOR], AX

        POP DI
        POP AX
        ADD DI, WORD[RESERVED_SECTORS]
        ADD DI, AX

        MOV AX, WORD[EXPLORER_FIRST_SECTOR]
        MOV BX, DOS_OFFSET

READ_DOS:
        PUSH AX

        SUB AX, 2
        MOV CL, BYTE[SECTORS_PER_CLUSTER]
        XOR CH, CH
        MUL CX

        ADD AX, DI
        MOV CL, BYTE[SECTORS_PER_CLUSTER]
        MOV DL, BYTE[DRIVE_NUMBER]
        MOV SI, DOS_SEGMENT
        MOV ES, SI
        CALL READ_DISK
        JC ERROR

        MOV AL, CL
        XOR AH, AH
        MUL WORD[BYTES_PER_SECTOR]

        ADD BX, AX

        POP AX
        PUSH BX

        MOV BX, FILESYSTEM >> 4
        MOV ES, BX

        MOV BX, AX
        SHR BX, 1
        ADD BX, AX
```

```
        TEST AX, 1
        MOV AX, WORD[ES:BX]
        JZ EVEN_CLUSTER

        ; SHR AX, 4
        SHR AX, 1
        SHR AX, 1
        SHR AX, 1
        SHR AX, 1
        JMP ODD_CLUSTER

EVEN_CLUSTER:
        AND AX, 0x0FFF

ODD_CLUSTER:
        POP BX

        CMP AX, 0xFF8
        JL READ_DOS

        MOV DX, WORD[ROOT_DIR_LOC]

        MOV SI, DOS_SEGMENT
        MOV DS, SI

        JMP DOS_SEGMENT:DOS_OFFSET

ERROR:
        MOV SI, ERROR_MSG ; Print the error message.
        CALL PUTS

        MOV AH, 0x01
        INT 0x13

        MOV BL, AH
        CALL PUTH8

        MOV AH, 0x00
        INT 0x16 ; Wait for a key to be pressed.

        JMP 0xFFFF:0x0000 ; Reboot.

; AX <- LBA value.
; CL <- Number of sectors to read.
; DL <- Drive number.
; ES:BX <- Pointer to buffer.
READ_DISK:
        PUSH AX
        PUSH BX
        PUSH CX
        PUSH DX
        PUSH SI
        PUSH DI
        PUSH ES

        XOR CH, CH
        MOV DI, CX

.READ_LOOP:
        CALL LBA_TO_CHS
        CALL READ_CHS
        JC .RETURN

        MOV SI, ES
        ADD SI, 32
        MOV ES, SI

        INC AX
        DEC DI
        JNZ .READ_LOOP

.RETURN:
        POP ES
        POP DI
        POP SI
        POP DX
        POP CX
        POP BX
        POP AX
        RET

; ES:BX <- Pointer to target buffer.
; CX[0:5] <- Sector number.
; CX[6:15] <- Track/Cylinder.
```

```
; DH <- Head number.
; DL <- Drive number.
READ_CHS:
       PUSH DI
       MOV DI, 3

.READ_LOOP:
       STC
       PUSH AX
       MOV AH, 0x02
       MOV AL, 1
       INT 0x13
       POP AX
       JNC .OUT

       DEC DI
       JNZ .READ_LOOP
       STC

.OUT:
       POP DI
       RET

; AX <- LBA value.
;
; CX[0:5] -> Sector number.
; CX[6:15] -> Track/Cylinder.
; DH -> Head number.
LBA_TO_CHS:
       PUSH AX
       PUSH DX

       XOR DX, DX
       DIV WORD[SECTORS_PER_TRACK]
       INC DX
       MOV CL, DL ; Get the sector number.

       XOR DX, DX
       DIV WORD[HEAD_COUNT]
       MOV DH, DL ; Get the head number.

       MOV CH, AL
       ; SHL AH, 6
       ROR AH, 1
       ROR AH, 1
       AND AH, 0xC0
       OR CL, AH ; Get the number of tracks/cylinders.

       POP AX
       MOV DL, AL
       POP AX
       RET

; SI <- Pointer to string.
PUTS:
       PUSH AX
       PUSH BX
       PUSH SI

.LOOP:
       LODSB
       TEST AL, AL
       JZ SHORT .RETURN
       MOV AH, 0x0E
       MOV BX, 7
       INT 0x10
       JMP SHORT .LOOP

.RETURN:
       POP SI
       POP BX
       POP AX
       RET

; BL <- Value to print.
PUTH8:
       PUSH AX
       PUSH CX
       MOV AH, 0x0E
       MOV CX, 2

.LOOP:
       ; ROL BL, 4
       ROL BL, 1
       ROL BL, 1
```

```
        ROL BL, 1
        ROL BL, 1
        MOV AL, BL
        AND AL, 0x0F

        CMP AL, 10
        SBB AL, 0x69
        DAS

        INT 0x10

        DEC CX
        JNZ .LOOP

        POP CX
        POP AX
        RET

EXPLORER_FIRST_SECTOR: DW 0
ROOT_DIR_LOC: DW 0

LOADING_MSG: DB "Loading...", 0x0A, 0x0D, 0x00
ERROR_MSG: DB "Error -> 0x", 0x00

TIMES 510 - ($ - $$) DB 0
DW 0xAA55
```