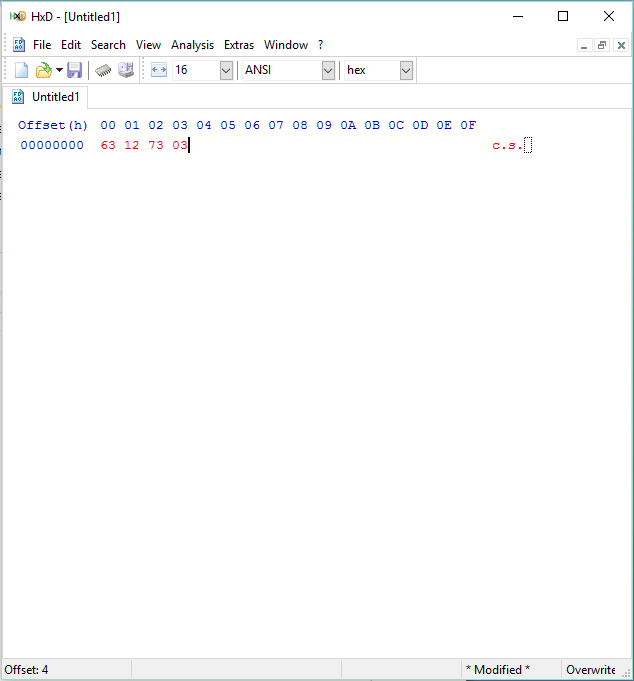
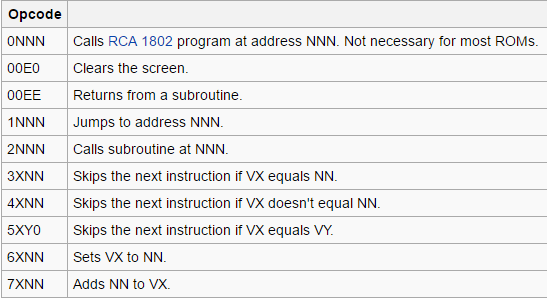
**Chip 8 Testing**

**Testing av opkoder**

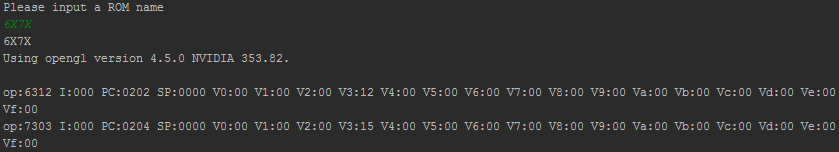
For å teste alle operasjonskodene våre utledet vi individuelle tester for hver eneste en og til slutt laget tester som testet dem i grupper. For å gjøre dette lagde vi test ROMs i en hexadecimal editor. Et slikt redigeringsprogram lar oss skrive inn heksaadesimal input som vi da kan lagre som en ROM fil og laste inn i ROM loaderen vår. Når loaderen kjører filen så leser den gjennom den heksadesimale inputen som programmet da tolker som forskjellige opkoder den skal gjøre noe med.   
  


*(Figur X.1) HxD Hovedmeny. HxD er et gratisprogram utviklet av* [*MH-Nexus*](https://mh-nexus.de/en)

Vi kan ta hex-verdiene fra figur X.1 og kjøre dem gjennom ROM loaderen. Det første vi vil vite er jo hvilken opkoder som kjører og hva de gjør. Opkoder er identifisert i grupper av 4. Dette vil si at ”63 12” er en operasjon, og 73 03 en annen. For å finne ut hvilken opkode som blir kjørt og hva den gjør så tar vi en titt på figuren under.

  
*(Figur X.2) En liten del av opkode tabellen fra* [*Chip8*](https://en.wikipedia.org/wiki/CHIP-8) *sin Wikipedia side.*

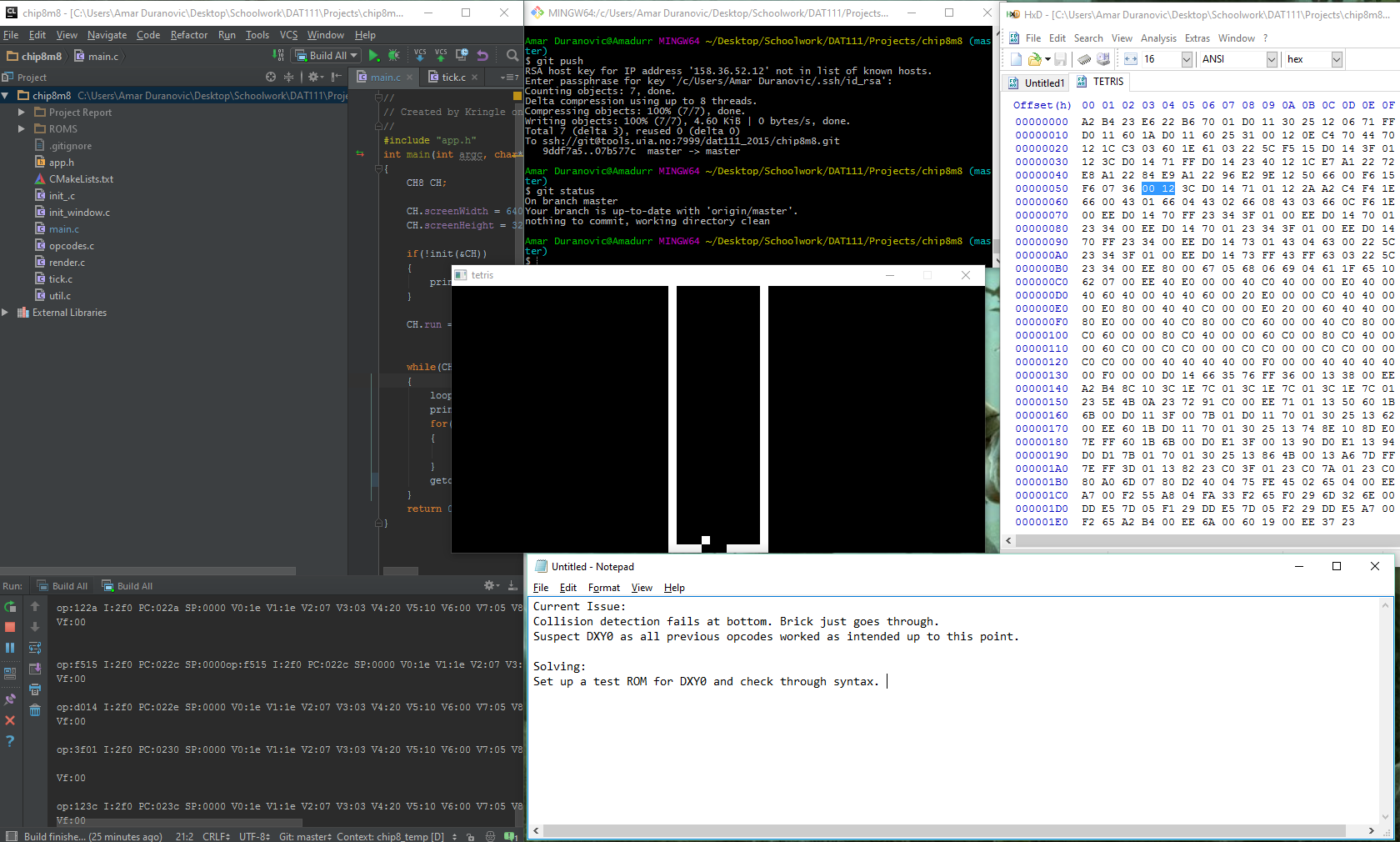
Det første sifferet identifiserer hvilken opkode som skal kjøres. I det første tilfellet ’6’ så sier det oss at det er opkode 6XNN som kjøres. For de 3 neste siffrene trenger vi bare følge beskrivelsen til opkoden i tabellen. I dette tilfellet blir desimalverdien 18 (12 er den heksadesimale verdien, som da gir oss 18 desimalt) lagt inn i register nummer 3. Etter denne operasjonen er utført blir opkode 7XNN utført. Den legger verdien 3 til register V3. Dette betyr at det nå burde ligge en total av 21 på register V3. Det sjekker vi i konsollen:



*(Figur X.3) Konsollen til programmet.*

Ved første operasjonskode ser vi at den kjører 6312 som planlagt og setter 12 (Heksadesimalt) inn i V3. Da vet vi at den fungerer som den skal i dette tilfellet. Etterpå ser vi at op:7303 kjøres og den legger til 3 i V3, som gir en total av 15 (21 desimalt). Slik ser vi at opkode 6XNN og 7XNN fungerer slik som de skal i dette tilfellet.

**Feilsøking**

*******(Figur X.4) Jakten på bugs.*

Når man utvikler noe så er man så og si garantert for å komme borti feil. I figur X.4 ser vi en ganske morsom feil vi fikk når vi prøvde å kjøre tetris. Den tegna opp banen og slapp brikken uten problemer, men når brikken kom ned til bunnen gikk den bare gjennom banen å stoppet programmet. Etter en god latter satte vi oss ned og analyserte problemet. Vi steppet programmet frem til hvor feilen skjedde, fant ut hvilke opkoder som ble brukt og så sjekket dem individuelt. Vi lærte at når feil oppstår så er det viktig og sette seg ned å få et perspektiv over problemet. Når man har en ide om hva som kan være feil setter man i gang med prosessen som er nødvendig for å fikse disse feilene. Nedenfor er en tabell for alle testene vi utførte på de forskjellige operasjonskodene.

|  |  |  |
| --- | --- | --- |
| Operasjonskode | Funksjon | Test |
| 00E0 | Renser skjermen | Tegner tilfeldige prikker på skjermen, kaller 00E0, ser om skjermen er klarert (Blank) |
| 00EE | Returnerer fra en sub-rutine | Kaller sub-rutinen, kaller 00EE og sjekker data. Kaller 00EE igjen og sjekker for underflow. |
| 1NNN | Hopper til adressen NNN | Gjør flere hopp mellom adresser, itererer V0 for hvert hopp, sjekker om V0 er rett på slutten. |
| 2NNN | Kaller sub-rutinen på NNN | Kaller sub-rutinen, sjekker om alt er i orden, kaller inn 16 til og sjekker for overflow. |
| 3XNN | Hopper over neste instruksjon om VX er lik NN | Skriver NN (f.eks 10) til et register (f.eks V5) og kaller 3XNN (3510). Om de ikke er like, skriv 0xFF til et register og kjør 3XNN+1 (3511). Om de er like, skriv FF til et annet register. Kjør til slutt 3XNN-1 (350F). Om like skriv 0xFF til et annet register, så utfør et register dump og sjekk hvor den mislyktes. |
| 4XNN | Hopper over neste instruksjon om VX IKKE er lik NN | Veldig likt som i 3NNN. Snu på lik/ikke lik lokasjonene. |
| 5XY0 | Hopper over neste instruksjon om VX er lik VY | Skriv NN-1 til et register, NN til neste, NN+1 på neste og NN til det siste. Sammenlign de tre første med den siste, skriv 0xFF til registere som mislykkes. |
| 6XNN | Setter VX til NN | Skriver NN til alle registere, sjekker data så overskriver et register flere ganger, sjekker alltid etter feil. |
| 7XNN | Setter NN til VX | Laster inn 1 til et register, adderer 1 til det registeret, fortsetter med dette og holder tellingen. Sjekker om første register = 0x00, om ikke, loop tilbake til den første addisjonen, sjekk data hvis lik. |
| 8XY0 | Setter VX til verdien av VY | Mye likt som 6XNN, men skriver til alle registrene fra et, så til et register fra alle. |
| 8XY1 | Setter VX til VX ELLER VY (Logisk ELLER) | Skriv 55 til et register og 0xAA til et annet. Kjør 8XY2, sjekk om vi får 0xFF fra VX men at VY enda er 0xAA. Gjenta for 0x5A, 0xA5 og til slutt 0x5A og 0x5A. |
| 8XY2 | Setter VX til VX OG VY (Logisk OG) | Likt som 8XY1, men med AND logikk. |
| 8XY3 | Setter VX til VX XELLER VY (Logisk XELLER) | Likt som 8XY1, men med XOR logikk. |
| 8XY4 | Adderer VY til VX. VF setter til 1 om det er et carry og til 0 hvis ikke. | Skriv N1 til et register, og N2 til et annet. Kjør 8YX4, Sjekk om X = N1 + N2, hvis usann hopp til enden av programmet og dump registrene. Hvis sann kjør 8XY4, og sjekk om Y = N2+(N1+N2). (Vær sikker på at tallene ikke skaper overflow). Skriv FF til register Z og kjør 8YZ4 og sjekk om Y = (N2+(N1+N2))-1(Pga binærlogikk) OG at VG = 0x01. |
| 8XY5 | VY substraheres fra VX. VF settes til 0 om det er et mente og 1 om det ikke er. | Skriv N1 til et register så N2 til et annet. Kjør 8XY45, sjekk om X = N1-N2, hvis usant hopp til slutten av programmet og dump registrene. Hvis sant kjør 8YX5 og sjekk om Y = N2-(N1-N2)(Vær sikker på å velge tall som ikke skaper underflow) Skriv 0xFF til register Z og kjør 8YZ5 og sjekk om Y = (N2-(N1-N2))+1(På grunn av binærlogikk) OG at VF = 0x00. |
| 8XY6 | Bitskifter VX til høyre med en. VF settes til verdien av det minst signifikante bitet av VX før skiftet. | Skriv 0x80 til et register, så bruk 8XY6 på det registeret og legg til 0x01 til et annet. Hvis VF != 1 loop tilbake til skiftet, hvis VF=1 sjekk data. |
| 8XY7 | Setter VX til VY minus VX. VF settes til 0 om det er et mente og 1 om det ikke er mente. | Likt som testen for 8XY5, bortsett fra at man fortsetter med det andre registeret. |
| 8XYE | Bitskifter VX til venstre med en. VF settes til verdien av det mest signifikante bitet av VX før skiftet. | Likt som testen for 8XY6, men skriv først 01 til registeret. |
| 9XY0 | Hopper over den neste instruksjonen om VX ikke er likt VY. | Likt som 5XY0 men skift ut de like med ulike og motsatt. |
| ANNN | Setter I til adressen NNN | Setter ANNN til 0x000, sjekker data, så med FFF, så 0xA5A og så 0x5A5. |
| BNNN | Hopper til adressen NNN pluss V0 | Hopp rundt et par ganger mens et register blir iterert. Sjekk data på slutten, hvis registeret ikke har rett verdi så har det gjort et feilhopp. |
| CXNN | Setter VX til resultatet av en bitvis-OG operasjon mellom et tilfeldig tall og NN | Kall CXNN et par ganger og skriv til neste register hver gang. Sjekk data og kjør hele prosessen et par ganger for å se om mønsteret gjentar seg. |
| DXYN | Sprites lagret i minne på lokasjon i indeks registeret (I), 8 bit brede. Pakker inn rundt skjermen. Hvis den tegner, renser en piksel, register VF setttes til 1 ellers er den 0. All tegning er XELLER tegning (m.a.o den veksler mellom skjerm pikslene). Sprites blir tegner i posisjon VX, VY. N er antall 8 bit rekker som må tegnes. Hvis N er større en 1, fortserrer andre linjen på posisjon VX, VY+1, og så videre | Skriv sprites til tilfeldige steder en stund, clear skjermen og tegn i en loop langs bunnen for å teste for overflows. |
| EX9E | Hopper over neste instruksjon hvis knappen som er lagret i VX er trykket ned | Tildel knapp 0 til VX så kjør EX9E i en loop som avslutter når knappen blir trykket. Iterer VX og gjenta til VX >= 0x0F |
| EXA1 | Hopper over neste instruksjon hvis knappen som er lagret i VX ikke er trykked ned | Samme som EX9E (Logikk til å hoppe over (Gitt EXA1 er på adressen 0x200)EXA1 1206 1200) |
| FX07 | Setter VX til verdien av delay timeren. (DT) | Sett DT så kjør FX07 og se i deconstructoren om VX == DT. Gjenta et par ganger. |
| FX0A | Et knappetrykk ventes og blir så lagret i VX. | Printer den trykte knappen på skjermen, venter på hvilken som helst knapp så renser skjermen og gjentar. |
| FX15 | Setter delay timeren (DT) til VX | Skriv 0x3C(Hexkode for 60, DT kjører på 60hz) til et register og kopier det til DT. Skriv DT til et register og sjekk om det er 0. Gjenta til det er 0 så avslutt. Sammenlign dette med en stoppeklokke eller timer manuelt for å se om den bruker cirka et sekund. |
| FX18 | Setter sound timeren (ST) til VX | Mye likt som FX15 men uten en sjekk loop siden den vil stoppe av seg selv. Bare sammenlign det med en stoppeklokke for å se hvor lenge den går. |
| FX1E | Adderer VX til I | Skriv noe satt data til et par registere og sett dem til I. Skriv 0xFF til V0 så hopp. Hvis den hopper til det rette stedet setter den V0 til 0x00 og hopper til slutten. Hvis ikke så hopper den bare til slutten eller kræsjer. |
| FX29 | Setter I til lokasjonen av spriten for karakteren i VX. Karakterer 0-F (Heksadesimalt) er representert av en 4x5 font | Kunne vært samme som FX0a, men anderledes feil kunne maskert hverandre. Så test denne på egenhånd SÅ test FX0A. Skriv tall fra 0x00 til 0x0F til de samsvarende registrene, så kjøre hver eneste med FX29 og printe de på skjermen på siden av hverandre. |
| FX33 | Lagrer den binærkodete desimal representasjonen av VX med de mest signifikante av de tre siffrene i adressen I. Det midterste sifferet på I +1, minst signifikante sifferes på I + 2. (m.a.o, tar desimal representasjonen av VX, plasserer hundredels sifferet i minnelokasjonen i I, tierene i lokasjonen I+1 og enerene i lokasjon I+2) | Skriv et random tall til VC og skriv tallene av det minnet med begynnelsen i 0x300. Iterer I med 3 og gjenta, last så opp 8 registere fra minne med begynnelse i 0x300. |
| FX55 | Lagrer V0 til VX (Inkluderer VX) i minne som begynner med adresse I | Skriv indeksene av hvert register til dem selv og skriv dette til minnet. Last registrene med tilfeldig data og last så tilbake til registrene fra minnet. |
| FX65 | Fyller V0 til VX (Inkluderende VX) med verdier fra minne som begynner med adresse I | Star en test med et hopp, så 00, 01, 02... 0E 0F og sett I til 202 og kjør FF65. |