

1 Wersje konsol

Pierwotnym wydaniem konsoli był Family Computer z 1983 roku, w skróconej formie nazywany "Famicom". Famicom oficjalnie nigdy nie został wydany poza Japonią. Nintendo przygotowując się do wydania międzynarodowego swojej konsoli do grania podjęło decyzję o przeprojektowaniu wyglądu sprzętu w celu dostosowania się do zachodnich odbiorców. Wynikiem tych prac był Nintendo Entertainment System, wydany w 1985 roku w USA, rok później na terenie Europy oraz w 1987 roku w pozostałych lokalizacjach w Europie oraz Australii[?].



Rysunek 1: Family Computer wraz z kontrolerami[?]

Rysunek 2: Nintendo Entertainment System NES-001 wraz z kontrolerem[?]

Odświeżone wersje konsol zostały wydane w 1993 roku. Nintendo znacznie zunifikowało wygląd zewnętrznego konsoli, zachowując kompatybilność kartridzy dla poszczególnych regionów.



Rysunek 3: "New Famicom" wraz z kontrolerem[?] kontrolerem[?]

Rysunek 4: Nintendo Entertainment System NES-101 wraz z

Obok oryginalnych sprzętów od Nintendo równocześnie egzystowały konsole podrabiane na masową skalę, w większości były to produkty kompatybilne z japońskim Famicomem. Dokładna historia nie jest znana; zebrane informacje pozwalają na stwierdzenie, iż pierwsze podróbki zaczęły pojawiać się pod koniec lat osiemdziesiątych oraz zostały wyprodukowane na Tajwanie. W latach dziewięćdziesiątych następowała miniutaryzacja sprzętów oraz minimalizacja kosztów, co spowodowało stopniowo powiększane przenoszenie produkcji do Chin oraz drastyczny spadek jakości wykonania konsol. Kreatywność "piratów" doprowadziła do powstania sprzętów będących czymś innym niż zwyczajna konsola:

- Konsole z dwoma gniazdami na kartridże - jedno dla gier z Famicoma, drugie dla gier z NES,
- Konsole z dwoma gniazdami na kartridże - jedno standardowo na zewnątrz obudowy, drugie wewnątrz obudowy z fabrycznie zamontowaną grą. Dzięki temu rozwiązaniu gdy w zewnętrznym gnieździe nie było zamontowanego kartridża to uruchamiał się kartridż wbudowany w konsolę,
- Konsola-klawiatura, udająca komputer - zawsze dołączano kartridż z "programami multimedialnymi" które korzystały z klawiatury. Była tu pewna inspiracja oficjalnym Family Basic,
- Przenośna konsola z wbudowanym wyświetlaczem,
- Sprzęty "Plug 'n' Play" o wymyślnych kształtach, często były one pozbawione gniazda na kartridże, zatem wbudowane gry musiały wystarczyć.



Rysunek 5: Generation NEX - klon z dwoma gniazdami na kartridże, górne jest dla gier Famicom, boczne dla gier NES[?]



Rysunek 6: Daryar DY-400-656 - konsoła z wbudowaną grą 400 in 1[?]



Rysunek 7: GLK-2004 - popularna konsola-klawiatura[?]



Rysunek 8: Game Axe Color - jeden z pierwszych handheldów (2000 rok) odwierający gry z Famicoma[?]

Klony konsoli Nintendo są produkowane po dzień dzisiejszy, tyle że częściowo wyszły ze szarej strefy - "Wielkie N" nie ma podstaw prawnych do sądzenia się z firmami gdyż wygasły patenty (TODO: sprawdzić dokładnie jak to wygląda, lecę z głowy z tego co kiedyś czytałem). Konstrukcje są również uwspółczesniane poprzez zastosowanie gniazda HDMI czy obsługę gier z kilku platform jednocześnie.



Rysunek 9: RetroUSB AVS - konsola wysokiej jakości do której stworzenia użyto podzespołów z oryginalnych NES[?]



Rysunek 10: Hyperkin RetroN 5 - sprzęt obsługujący ogromną ilość platform jednocześnie, również kartridże do NES i Famicoma[?]



Rysunek 11: NES 620 Games - niskobudżetowy, współczesny klon konsoli NES z wbudowanymi grami, bez zewnętrznego gniazda na kartridże[?]

Nieoficjalne sprzęty zyskały popularność w biedniejszych krajach w których Nintendo nie dystrybuowało swoich produktów. "Marki" konsol różniły

się między poszczególnymi państwami, niektóre kraje miały nawet "oficjalnych dystrybutorów tychże podróbek. Miały również miejsce sytuacje, gdzie jeden i ten sam model / typ konsoli był sprzedawany pod kilkoma nazwami.[?]



Rysunek 12: Micro Genius IQ-502 - wersja konsoli sygnowana marką producenta[?]



Rysunek 13: Pegasus IQ-502 - wersja znaczona na rynek rosyjski, Dendy to Micro Genius IQ-502 przeznaczona na rynek polski, Pegasus to marka firmy Bobmark.[?]

Wszystkie trzy powyższe sprzęty to w rzeczywistości jeden projekt firmy Micro Genius, różniący się logotypami na konsoli oraz padach.

2 Specyfikacja konsoli

Procesor (CPU): Ricoh 2A03 (NTSC) / Ricoh 2A07 (PAL), ośmiorozmiotowy mikroprocesor będący modyfikacją MOS 6502. Różnice 2A03 / 2A07 względem produktu MOS Technology są następujące[?]:

- Deaktywowany tryb dziesiętny (BCD),
- Wbudowany generator dźwiękowy,
- Obsługa kontrolerów poprzez porty \$4016 oraz \$4017,
- DMA.

Mimo iż jest to 8-bitowy procesor, można zaadresować 64 kilobajty pamięci. To, jak podzielona jest ta przestrzeń adresowa przedstawia poniższa tabela (podane wartości są zapisane w systemie szesnastkowym):

Adresy	Rozmiar	Zastosowanie
\$0000 - \$07FF	\$0800	Wewnętrzna pamięć RAM (dwa kilobajty)
\$0800 - \$1FFF	\$0800	Duble danych spod adresów \$0000 - \$07FF (co dwa kilobajty)
\$2000 - \$2007	\$0008	Rejestry kontrolujące PPU
\$2008 - \$3FFF	\$1FF8	Duble danych spod adresów \$2000 - \$2007 (co osiem bajtów)
\$4000 - \$4017	\$0018	Rejestry kontrolujące APU oraz rejesty wejścia / wyjścia
\$4018 - \$401F	\$0008	Dezaktywowany tryb samotestowania
\$4020 - \$FFFF	\$BFE0	Przestrzeń do dowolnej dyspozycji dla kartridża

Adresy \$4020 - \$FFFF są zaadresowane zgodnie z możliwościami kartridża (patrz Mapper) [?]

Układ graficzny (PPU): Ricoh 2C02, ośmiorozmiotowy układ opracowany przez Nintendo specjalnie dla tej konsoli. Mimo iż jest to 8-bitowy procesor, można zaadresować 16 kilobajtów pamięci. To, jak podzielona jest ta przestrzeń adresowa przedstawia poniższa tabela (podane wartości są zapisane w systemie szesnastkowym):

Adresy	Rozmiar	Zastosowanie
\$0000 - \$0FFF	\$1000	Pattern Table 0
\$1000 - \$1FFF	\$1000	Pattern Table 1
\$2000 - \$23FF	\$0400	Nametable 0
\$2400 - \$27FF	\$0400	Nametable 1
\$2800 - \$2BFF	\$0400	Nametable 2
\$2C00 - \$2FFF	\$0400	Nametable 3
\$3000 - \$3EFF	\$0F00	Dubel danych spod adresów \$2000 - \$2EFF
\$3F00 - \$3F1F	\$0020	Paleta barw
\$3F20 - \$3FFF	\$00E0	Duble palety barw (co 32 bajty)

Dostęp do wyżej wymienionych adresów odbywa się poprzez adresy CPU: \$2006 i \$2007. Gniazdo kartridzy: 60-pin (Famicom), 72-pin (NES)

2.1 Rejestry kontrolujące PPU

Adresy \$2000 - \$2007 oraz \$4014 w CPU mają specjalne znaczenie - poprzez nie programista / program może wpływać na działanie układu graficznego konsoli. Zwyczajowe nazwy tych adresów przedstawia poniższa tabela [?]:

Adres hex	Nazwa
\$2000	PPUCTRL
\$2001	PPUMASK
\$2002	PPUSTATUS
\$2003	OAMADDR
\$2004	OAMDATA
\$2005	PPUSCROLL
\$2006	PPUADDR
\$2007	PPUDATA
\$4014	OAMDMA

Nazw tych powszechnie używa się w kodzie programów zamiast adresów bezpośrednich oraz w dyskusjach na temat programowania konsol NES / Famicom. Taką zasadę stosuję również w tym dokumencie.

2.2 Rejestry kontrolujące APU

Adresy \$4000 - \$4013, \$4015 oraz \$4017 w CPU mają specjalne znaczenie - poprzez nie programista / program może wpływać na działanie układu

dźwiękowego konsoli. Zwyczajowe nazwy tych adresów przedstawia poniższa tabela [?]:

Adres hex	Nazwa
\$4000	PL1_VOL
\$4001	PL1_SWEEP
\$4002	PL1_LO
\$4003	PL1_HI
\$4004	PL2_VOL
\$4005	PL2_SWEEP
\$4006	PL2_LO
\$4007	PL2_HI
\$4008	TRI_LINEAR
\$400A	TRI_LO
\$400B	TRI_HI
\$400C	NOISE_VOL
\$400E	NOISE_LO
\$400F	NOISE_HI
\$400E	DMC_FREQ
\$400F	DMC_RAW
\$4010	DMC_START
\$4011	DMC_LEN
\$4012	SND_CHN

Nazwy te zostały zaczerpnięte z biblioteki audio do NES / Famicom - FamiTone2 [?]. Używam ich również w kodzie źródłowym programu oraz w tym dokumencie. (TODO: omówienie każdego rejestru, jeszcze nie kodowałem tego to nie znam szczegółów programowych)

3 Projekt

W tej sekcji omówię krok po kroku jak powstawał mój własny projekt programu na platformę NES. Zacznę od pojedynczych, prostych czynności aż po pewnego rodzaju pełnoprawny szkielet takiego projektu zgodny z dobrymi praktykami programowania NES. Następnie zostaną omówione krytyczne fragmenty kodu z punktu widzenia całego projektu oraz własne rozwiązania często spotykanych zagadnień.

3.0 Absolutne minimum

Kod w pełni inicjujący konsolę jest dość rozbudowany, zatem warto zacząć od czegoś łatwiejszego... Niech będzie to coś, co pokaże że jakkolwiek panujemy nad maszyną - zmiana wartości pojedynczej komórki pamięci to rozsądny pomysł. Stwórzmy plik źródłowy, nazwijmy go *main.s*. Kod ustalający wartość pierwszej komórki RAM na wartość 16 (w systemie szesnastkowym) będzie wyglądał następująco:

```
lda #$16  
sta $0000
```

Mnemonik *lda* ma wiele odmian - ta zastosowana powyżej wczytuje konkretną wartość do wbudowanego w CPU rejestru - akumulatora. Symbol *#\$* oznacza właśnie konkretną wartość liczbową zapisaną szesnastkowo. Mnemonik *sta* kopiuje wartość z akumulatora pod lokację podaną jako operand; tutaj jest to pierwsza komórka pamięci RAM (= o adresie \$0000).

Warto zrobić jeszcze jedną rzecz - dopisać za powyższymi poleceniami pętlę nieskończoną:

```
Stop:  
jmp Stop
```

Trzeba pamiętać o tym iż pod naszym programem nie ma żadnego systemu operacyjnego czy innego środowiska wykonywalnego, zanim nie ma mowy o zakończeniu wykonania programu - bez tej pętli CPU interpretowałby kolejne bajty jako kod, cokolwiek by tam było i wykonał je.

Jednakże jest to o wiele za mało żeby nawet emulator zrobił to, co oczekujemy. Pierwsza sprawa jest związana z ogólnie przyjętym formatem pliku reprezentującym ROM programu na platformę NES. Taki plik ma szesnastobajtowy nagłówek opisujący szczegóły programu. Nagłówek jest potrzebny emulatorom oraz flashcartom do przygotowania adekwatnego środowiska uruchomieniowego. W naszym przypadku nagłówek będzie wyglądał następująco:

```
.byte "NES"  
.byte $1A  
.byte 2  
.byte 0  
.byte %00000000  
.byte %00000000  
.byte 0, 0  
.byte 0, 0, 0, 0, 0, 0
```

Pierwsze trzy bajty ("NES") to magiczna wartość. Kolejny bajt to znak końca linii. Po wyżej opisanych czterech bajtach programy mogą rozpoznać iż mają do czynienia z plikiem w formacie *.NES. Bajt piąty to ilość 16KB stron pamięci przeznaczonych na kod wykonywalny. Dwa banki wypełniają całą przestrzeń adresową konsoli. Kolejny bajt to ilość 16KB stron pamięci przeznaczonych na grafikę. W tym momencie nie potrzebujemy żadnej grafiki zatem ustawiamy bajt na zero. Właściwie wartość zero ma inny efekt a my skorzystamy z efektu ubocznego tego ale jest to na razie nieistotne - uzyskamy pożądany efekt. Bajty 7 - 10 to flagi - na tę chwilę interesują nas tylko i wyłącznie bajty odpowiadające za Mapper. Mapper 0 jest "podstawowy" przez co odpowiedni do prostych programów, zatem go ustawimy. Warto wspomnieć o tym, jak formowany jest numer mappera: cztery górny bajty pochodzą z czterech górnych bajtów bajta ósmego w nagłówku, cztery dolne bajty pochodzą z czterech górnych bajtów bajta siódmego w nagłówku. (TODO: obrazek lepszy) Funkcjonalność bajtów 9 oraz 10 nie interesuje nas w tej chwili, można je ustawić na 0 i zapomnieć o ich istnieniu. Bajty 11 - 15 to dopełnienie nagłówka. Wartości mogą być dowolne, zwyyczajowo są to zera.

Druga sprawa ma związek z architekturą konsoli. Zawsze po uruchomieniu / zresetowaniu konsoli, pierwszą rzeczą jaką robi procesor to skok bezwzględny do adresu uformowanego z bajtów pod adresami \$FFFC (górny bajt) i \$FFFD (dolny bajt). W tym celu modyfikujemy kod w następujący sposób:

```
_INT_Reset:  
lda #$16  
sta $0000  
  
.word _INT_Reset
```

.word to dyrektywa asemblera działająca jak (.byte), tyle że podajemy wartości dwubajtowe zamiast jednobajtowych. _INT_Reset to etykieta, zostanie ona zamieniona na konkretny adres przez linker. Dwukropek definiuje etykietę; samą nazwę etykiety traktuje się jak konkretną wartość szesnastobitową. Nadal zostaje problem powiedzenia kompilatorowi oraz linkerowi że ma wstawić adres wyznaczony przez etykietę _INT_Reset dokładnie pod adres \$FFFC w ROM. Inna sprawa to wskazanie kompilatorowi / linkerowi iż nagłówek nie jest częścią kodu lecz ma się znaleźć na początku pliku reprezentującego ROM. Oba problemy rozwiąże dyrektywa asemblera .segment. Ukończony pierwszy program będzie wyglądał następująco:

```
.segment "HEADER"  
.byte "NES"
```

```

.byte $1A
.byte 2
.byte 0
.byte %00000000
.byte %00000000
.byte 0, 0
.byte 0, 0, 0, 0, 0, 0, 0

.segment "CODE"
_INT_Reset:
lda #$16
sta $0000

.segment "VECTORS"
.word _INT_Reset

```

.segment to dyrektywa mówiąca ”poniższa treść należy do bloku danych zdefiniowanego w pliku konfiguracyjnym linkera podanego jako argument dyrektywy”. Na tę chwilę brzmi to niezrozumiałe, bo jeszcze nawet nie wspominałem czym jest plik konfiguracyjny linkera. Jest to plik ze wskazówkami jak ma być przeprowadzony proces linkowania plików obiektowych. Do pakietu CC65 jest dołączony przykładowy plik dla platformy NES jednakże wymaga wielu zmian by być w zgodzie z dobrymi praktykami przyjętymi przez społeczność współcześnie programującą na tej konsoli, zatem nic nam po nim i trzeba samemu rozgryźć jak stworzyć własny. Stwórzmy nowy plik, nazwijmy go *nes.cfg*. W naszym przypadku plik będzie miał następującą treść:

```

MEMORY {
    RAM:      start = $0000,  size = $0800,  type = rw,  file = "";
    HDR:      start = $0000,  size = $0010,  type = ro,  file = %0,  fill = yes,  fill
    PRG:      start = $8000,  size = $8000,  type = ro,  file = %0,  fill = yes,  fill
}

SEGMENTS {
    BSS:      load = RAM,  type = bss;
    HEADER:   load = HDR,  type = ro;
    CODE:     load = PRG,  type = ro,  start = $8000;
    VECTORS:  load = PRG,  type = ro,  start = $FFFC;
}

```

/textit{MEMORY} oraz /textit{SEGMENTS} wyznaczają grupy zasad sterujących linkerem. Etykiety w sekcji /textit{SEGMENTS} opisują dwie rzeczy -

mapę pamięci w programie NES za pomocą atrybutu *start* oraz determinuje kolejność danych w pliku wynikowym dla etykiet z atrybutem */textittype* o wartości *ro*. Kolejność etykiet mówi jak zostaną ustawione dane w pliku wynikowym wyznaczone przez atrybut *load*. Wartości tam podane odpowiadają etykietom zdefiniowanych w sekcji *MEMORY*. Każda wpis definiuje wielkość obszaru, typ dostępu (odczyt / zapis), w jakim pliku wynikowym się znajdzie obszar, opcjonalne wypełnienie wartościami. Podsumowując: *SEGMENTS* odnosi się do reprezentacji programu na konsoli, *MEMORY* odnosi się do reprezentacji danych w pliku ROM.

Proszę zwrócić uwagę, że nazwy segmentów użyte w pliku źródłowym programu pokrywają się z etykietami w sekcji *SEGMENTS* w pliku konfiguracyjnym. Z tego dla pliku ROM iż:

- na początku pliku pojawi się szesnastobajtowy nagłówek,
- następne szesnaście kilobajtów zajmą dane spod segmentu *CODE*, pozostała wolna przestrzeń zostanie wypełniona zerami
- potem to, co jest pod segmentem *VECTORS* posłuży linkerowi do nadpisania ostatnich 4 bajty segmentu *CODE* ($\$8000 - \$FFFC = \$04$)

A dla konsoli wynika że:

- to co znajduje się za dyrektywą *CODE* w pliku źródłowym zostanie zamapowane na obszar $8000\text{--}FFFF$ na konsoli NES,
- to co znajduje się za dyrektywą *VECTORS* w pliku źródłowym zostanie zamapowane na obszar $FFFC\text{--}FFFF$ na konsoli NES, częściowo nadpisując segment *CODE*; jest to jak najbardziej pożądane.

Zanim potwierdzimy powyższe słowa, potrzebujemy stworzyć wynikowy plik ROM. Umieszczamy pliki ca65.exe oraz ld65.exe w tym folderze co nasz kod źródłowy oraz plik konfiguracyjny. Najpierw komplikacja:

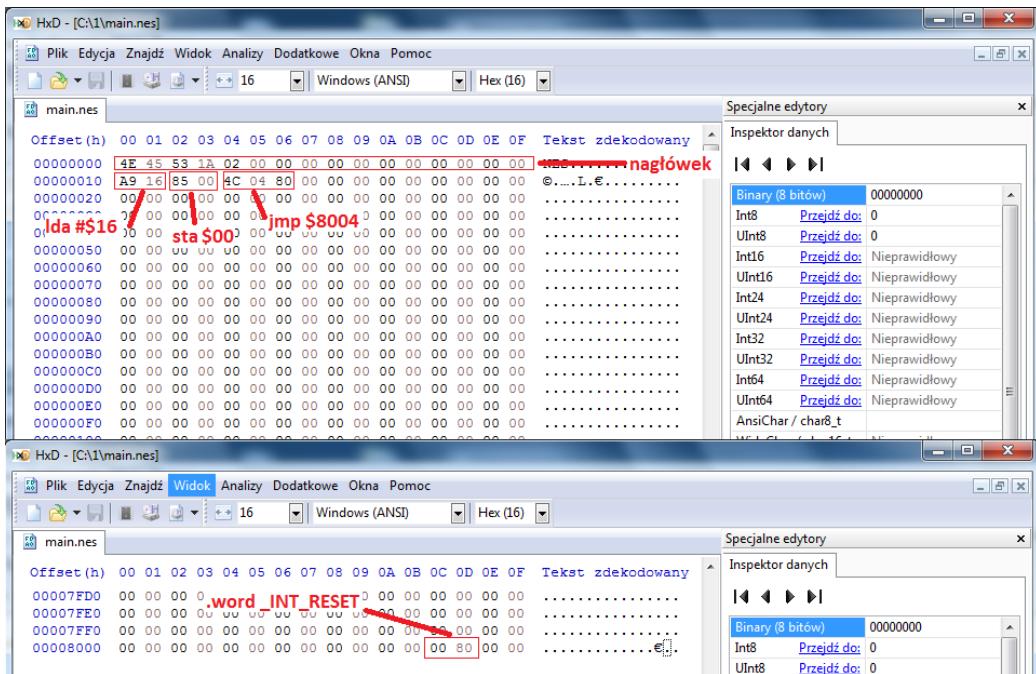
```
ca65 -o main.o main.s
```

Argument *-o* pozwala ustalić wynikową nazwę pliku obiektowego. Na końcu podajemy listę plików źródłowych. Następnie czas na linkowanie:

```
ld65 -t nes -o main.nes main.o
```

Argument *-t* ustala nazwę pliku, pod którą linker znajdzie konfigurację. Co ważne, plik musi mieć rozszerzenie *.cfg którego nie podajemy w powyższym poleceniu. Argument *-o* ustala nazwę wynikową pliku ROM. Na końcu jest lista plików obiektowych.

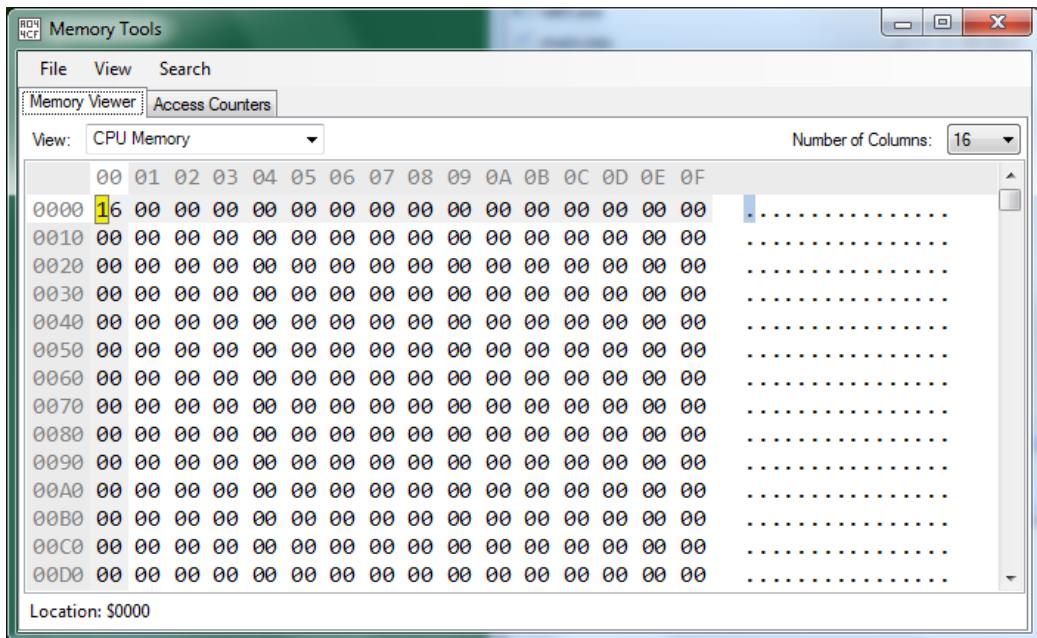
Zobaczmy jak te dwa narzędzia stworzyły plik ROM. Do tego trzeba posłużyć się oprogramowaniem zwanym hexedytorem:



Rysunek 15: Podgląd pliku ROM w programie HxD.

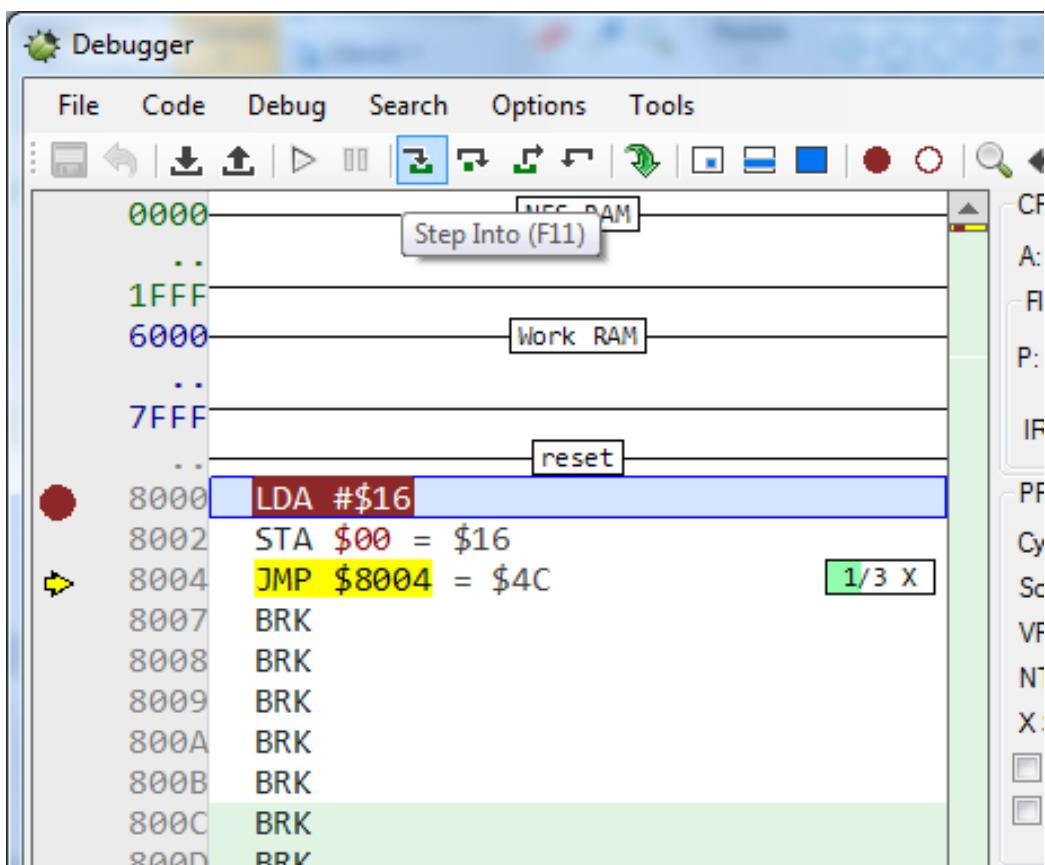
Kompilator inaczej zakodował instrukcję *sta* - związane jest to z tak zwanym Zero Page i nie ma wpływu na poprawność programu.

Teraz uruchommy ROM w Mesen. Przywita nas jednokolorowy ekran. Aby sprawdzić czy komórka pamięci zawiera pożdaną wartość, musimy wybrać z menu *Debug* opcję *Memory Tools* i upewnić się, że w dropdownie *View* mamy wybraną opcję *CPU Memory*:



Rysunek 16: Podgląd RAM w emulatorze Mesen.

Faktycznie, jest tam wartość szesnaście heksadecymalnie, zatem wszystko działa jak powinno. Jest to dobry przykład na naukę obsługi debuggera. Zatem w oknie głównym Mesena wybieramy z menu *Debug* opcję *Debugger*. Okno debuggera może przytłoczyć ogromem informacji, na początek skupimy się na podglądzie kodu. Postawmy breakpoint na adresie \$8000 (początek kodu) klikając LPM na lewo od numeru adresu. Następnie wyciągamy na wierzch okno z podglądem RAM aby widzieć naszą zmienianą komórkę pamięci a następnie z menu debuggera wybieramy *Debug* opcję *Reset*. Debugger sfokusuje się na adresie \$8000, co oznaczone jest żółtym tłem oraz żółtą strzałką wskazującą kolejną instrukcję do wykonania. Jednakże jest pewien problem - w pamięci już jest ustawiona wartość \$16. Otóż NES nie czyści pamięci podczas resetu - program sam musi o to zadbać.



Rysunek 17: Debugger w emulatorze Mesen.

Mimo tego prześledźmy wykonanie krok po kroku. W tym celu kilka razy kliknijmy przycisk *Step Into*. Otóż po wykonaniu instrukcji *sta* komórka w podglądzie RAM podświetliła się na czerwono - tak właśnie Mesen pokazuje, iż na danej komórce pamięci odbył się zapis.

4 Informacje o platformie

4.1 Zero Page

4.2 Mapper

4.3 Pattern Table

4.4 Nametable

4.5 Paleta barw

5 Narzędzia

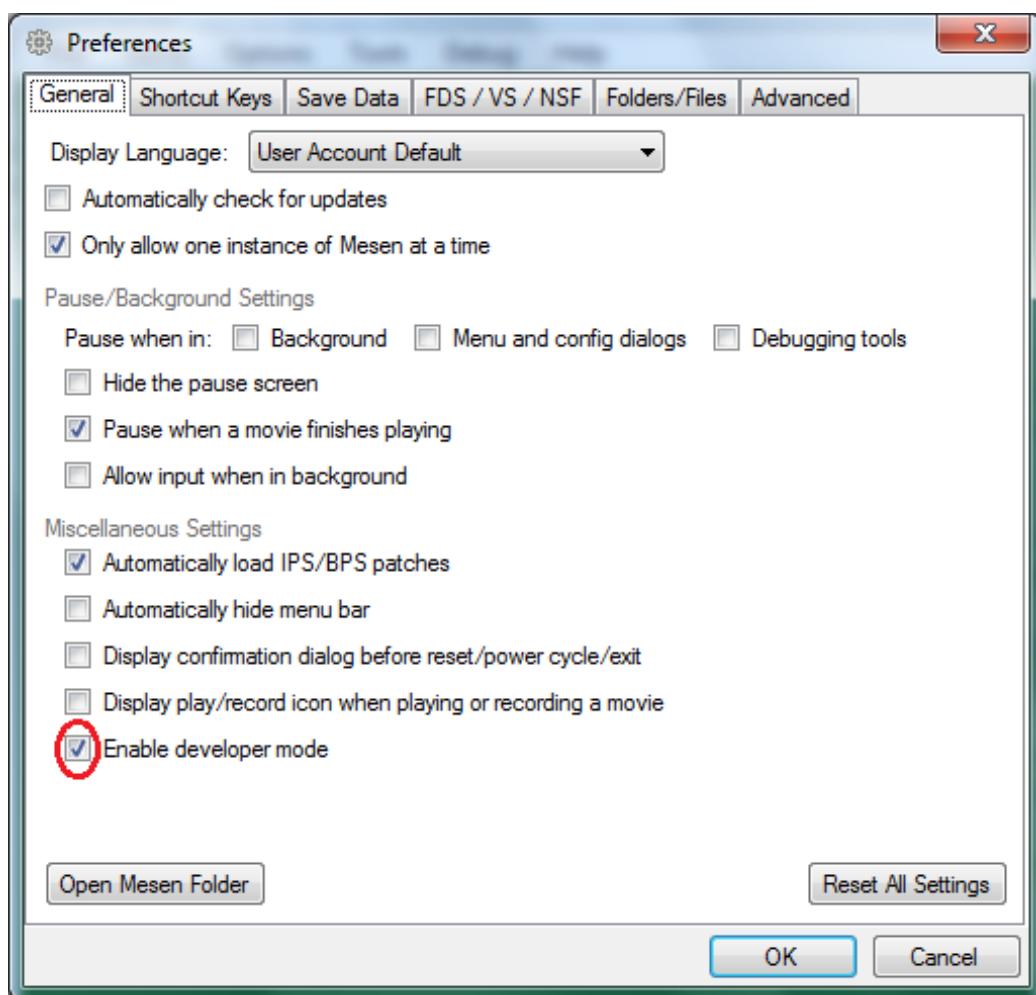
5.1 CA65

Strona projektu: <https://cc65.github.io/> CA65 jest częścią większego pakietu zwanego CC65. Jest on zbiorem narzędzi do komplikacji kodu napisanego w języku C na sprzętę z procesorem MOS 6502. Za to CA65 jest asemblerem pod ten sam procesor. W przypadku NES język C stanowi problem wydajnościowy, gdyż plik wynikowy który powstaje po komplikacji kodu w C jest znacznie wolniejszy niż odpowiednik asemblerowy; warto pamiętać iż ta konsola jest wielokrotnie mniej wydajna niż współczesne sprzęty. Stąd moja decyzja o pisaniu kodu tylko i wyłącznie w asemblerze.

(UWAGI: zestaw kompilator plus linker zasługuje na większe wyróżnienie od niżej opisanych programów pomocniczych do grafiki czy muzyki; do przemyślenia miejsce w dokumencie)

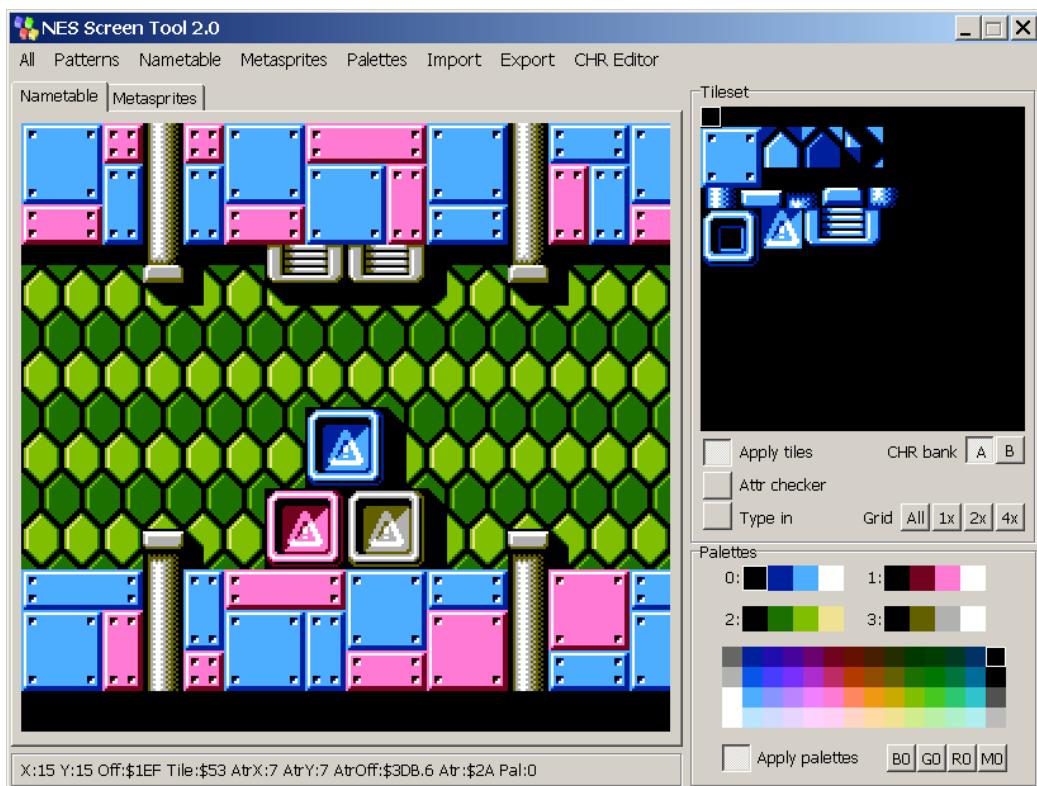
5.2 Mesen

Emulator znacząco wspiera proces tworzenia oprogramowania pod NES, gdyż zawiera wiele narzędzi pozwalających na zbadanie zachowania naszego programu. Aby mieć dostęp do narzędzi deweloperskich, musimy w menu *Options -> Preferences* zaznaczyć checkbox "Enable developer mode". Od tej pory w menu jest widoczna nowa pozycja - *debug*.



Rysunek 18: Przełącznik opcji deweloperskich w emulatorze Mesen.

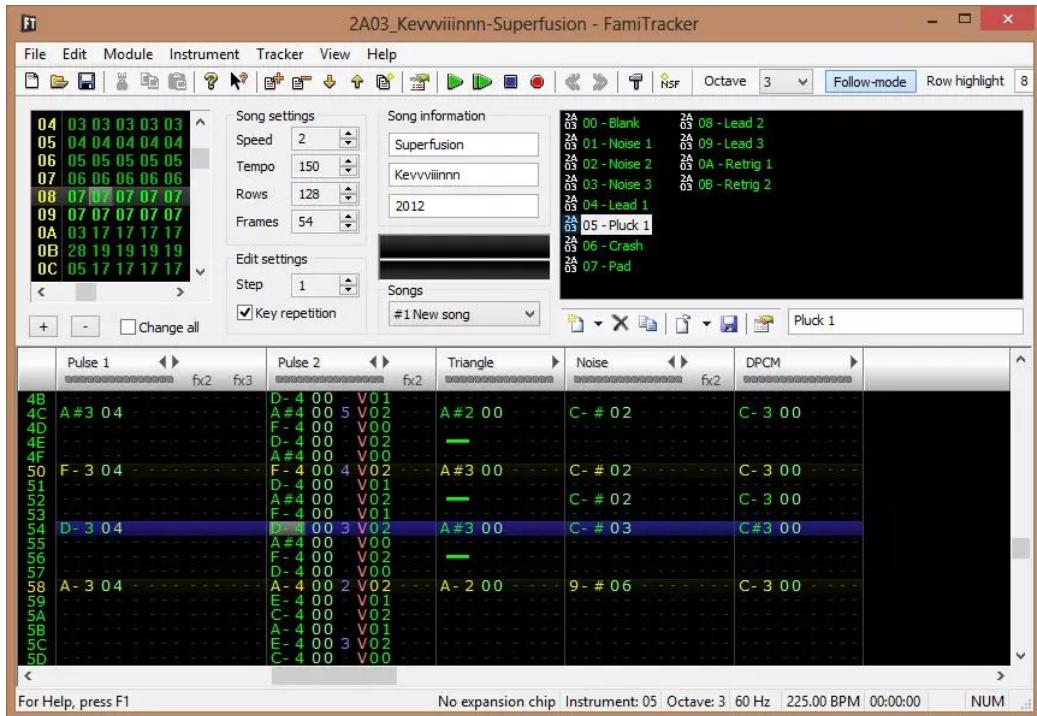
5.3 NES Screen Tool



Rysunek 19: Ekran główny aplikacji [?]

NES Screen Tool to program wspomagający tworzenie grafiki zgodnej z platformą NES / Famicom. Aplikacja pozwala na szczegółowe tworzenie grafik per pixel jak i bardziej ogólne działania jak eksport przygotowanych danych do formatu zrozumiałego przez konsolę, gotowego do zastosowania w kodzie programu. Strona domowa [?].

5.4 FamiTracker



Rysunek 20: Ekran główny aplikacji [?]

FamiTracker to program pozwalający na tworzenie muzyki w stu procentach kompatybilnej z platformami NES / Famicom. Aplikacja należy do rodziny trackerów - programów komputerowych w których komponuje się muzykę wykorzystując połączenie zapisu nutowego oraz poleceń sterujących efektami [?]. Strona domowa [?].

Literatura

- [1] <https://upload.wikimedia.org/wikipedia/commons/0/06/Nintendo-Famicom-Console-Set-FL.jpg>
- [2] <https://upload.wikimedia.org/wikipedia/commons/8/82/NES-Console-Set.jpg>
- [3] https://upload.wikimedia.org/wikipedia/commons/f/f1/New_Famicom.jpg
- [4] <https://upload.wikimedia.org/wikipedia/commons/e/e0/NES-101-Console-Set.jpg>

- [5] <https://levelupvideogames.com/used-messiah-generation-next-468309594161/>
- [6] <http://forum.pegasus-gry.com/index.php?topic=2315.0>
- [7] <https://arhn.eu/2018/04/historia-pegasusa-z-klawiatura-glk-2004-et-al/>
- [8] <https://www.ign.com/articles/2000/06/10/game-axe-color>
- [9] <https://upload.wikimedia.org/wikipedia/commons/7/7a/RetroUSB-AVS-Console-wController-FL.jpg>
- [10] https://cdn10.bigcommerce.com/s-2l8ru96d/products/223/images/1135/ret5bk_54484.1567
- [11] <https://www.amazon.com/Controllers-Children-Birthday-Childhood-Memories/dp/B083RBYRYN>
- [12] <http://www.retrogamingmuseum.com/the-collection/micro-genius-nes-clone>
- [13] <https://archiwum.allegro.pl/oferta/konsola-pegasus-iq-502-pady-pudelko-plomba-i7530854084.html>
- [14] <https://youla.ru/moskva/hobbi-razvlecheniya/konsoli-igry/dendy-classic-2-novaia-5b6c9ae1cf689a85567f44a6>
- [15] https://en.wikipedia.org/wiki/Nintendo_Entertainment_System
- [16] https://en.wikipedia.org/wiki/Nintendo_Entertainment_System_hardware_clone
- [17] Patrick Diskin: *Nintendo Entertainment System Documentation*, 2.1 2A03 Overview, <http://www.nesdev.com/NESDoc.pdf>
- [18] http://wiki.nesdev.com/w/index.php/CPU_memory_map
- [19] http://wiki.nesdev.com/w/index.php/PPU_registers#Summary
- [20] <http://wiki.nesdev.com/w/index.php/APU#Specification>
- [21] <http://forums.nesdev.com/viewtopic.php?t=7329>
- [22] <https://shiru.untergrund.net/software.shtml>
- [23] <https://shiru.untergrund.net/pic/nesst.png>
- [24] <http://famitracker.com/>

- [25] <https://blog.uptodown.com/wp-content/uploads/Famitracker-pianola.jpg.webp>
- [26] [https://pl.wikipedia.org/wiki/Tracker_\(oprogramowanie_muzyczne\)](https://pl.wikipedia.org/wiki/Tracker_(oprogramowanie_muzyczne))