C256 – Motorola 68000

Table of Contents

[Assemblers 1](#_Toc504671620)

[VASM 1](#_Toc504671621)

[DEVPAC3 1](#_Toc504671622)

[A68k 2](#_Toc504671623)

[Macro Assembler AS 2](#_Toc504671624)

[ASMX 2](#_Toc504671625)

[68KASM 2](#_Toc504671626)

[Monitors 2](#_Toc504671627)

[zBug V1.0 2](#_Toc504671628)

[Tutor 2](#_Toc504671629)

[Emuleren van de 68000 2](#_Toc504671630)

[Specifiek 68000 3](#_Toc504671631)

[Koude reset: 3](#_Toc504671632)

[ASCII to PETSCII conversion 4](#_Toc504671633)

[Standaard ASCII tabel 4](#_Toc504671634)

[PETSCII (C64) tabel 5](#_Toc504671635)

[Characters 0-127 charset 1 5](#_Toc504671636)

[Characters 128-255 charset 1 5](#_Toc504671637)

[Characters 0-127 charset 2 6](#_Toc504671638)

[Characters 128-255 charset 2 6](#_Toc504671639)

[Missende tekens uit standaard ascii 7](#_Toc504671640)

[Calling Conventions, Stack operations and Function Calls 8](#_Toc504671641)

[Opletten 8](#_Toc504671642)

# Assemblers

Visual Studio Code is een aardige editor, en kan een module installeren om 68000 assembly goed weer te geven (M68k-Assembly).

## VASM

Waarschijnlijk is VASM het beste wat er is op dit moment onder de cross assemblers (voor wat betreft m68k). Opletten met compilen: “make CPU=m68k SYNTAX=mot” geeft een juiste versie van de assembler (madmac nog onderzoeken?). Verder zorgt de optie “-Fbin” tijdens assembleren een flat binary output, geschikt om in te laden in een eigen emulator. Helaas niet zomaar commercieel gebruik toegestaan – dan contact opnemen met de ontwikkelaar.

## DEVPAC3

Een native (goede) assembler voor op de ST is trouwens DEVPAC3, en lijkt de de facto standaard geweest te zijn vroeger.

## A68k

Een variant van X68000 (Anderson, 1985, Dobb’s Magazine). Lijkt alleen gebruikt te worden voor TICALC toestanden. Het lukt me ook niet om flat binaries te produceren met deze assembler.

## Macro Assembler AS

Iets gemakkelijker commercieel te gebruiken dan VASM. Het lukt alleen niet om een fatsoenlijke binary te produceren. Mogelijk omdat het een ontwikkelings snapshot betreft. Zie website: <http://john.ccac.rwth-aachen.de:8000/as/>

## ASMX

Beste alternatief na VASM. De licentie lijkt alles toe te staan. Om de boel te compilen: Verander de Makefile in <src> directory; TARGET\_ARCH mag allemaal worden commented. Zie verder: <http://xi6.com/projects/asmx/>

Typische commandline: <<asmx -e -w -b -l test.lst test.asm>>

-e: show errors

-w: warnings

-b: produce binary

-l: produce listing

## 68KASM

Onderdeel van BSVC emulator pakket. Deze assembler is in C geschreven en stamt uit ongeveer 1986. Geen onderscheid tussen lower en upper case. Produceert Motorola s-records (\*.h68). Deze hex files kunnen worden omgezet in binaries met bijvoorbeeld SREC2BIN (s-record.com). Lijkt goed te werken met Musashi.

# Monitors

## zBug V1.0

## Tutor

The Tutor monitor program from the original Motorola Education Computer board modified for EASy68K. Source code is beschikbaar.

# Emuleren van de 68000

Wat er te vinden is op internet is lastig om iets mee te doen. Beste lijkt nog Musashi te zijn (heeft de cpu core geleverd voor MAME). Als eerst moet een soort m68kmake.c ding gebouwd worden (er staat een main functie in) en gerund. Deze maakt extra source files aan met opcodes etcetera.

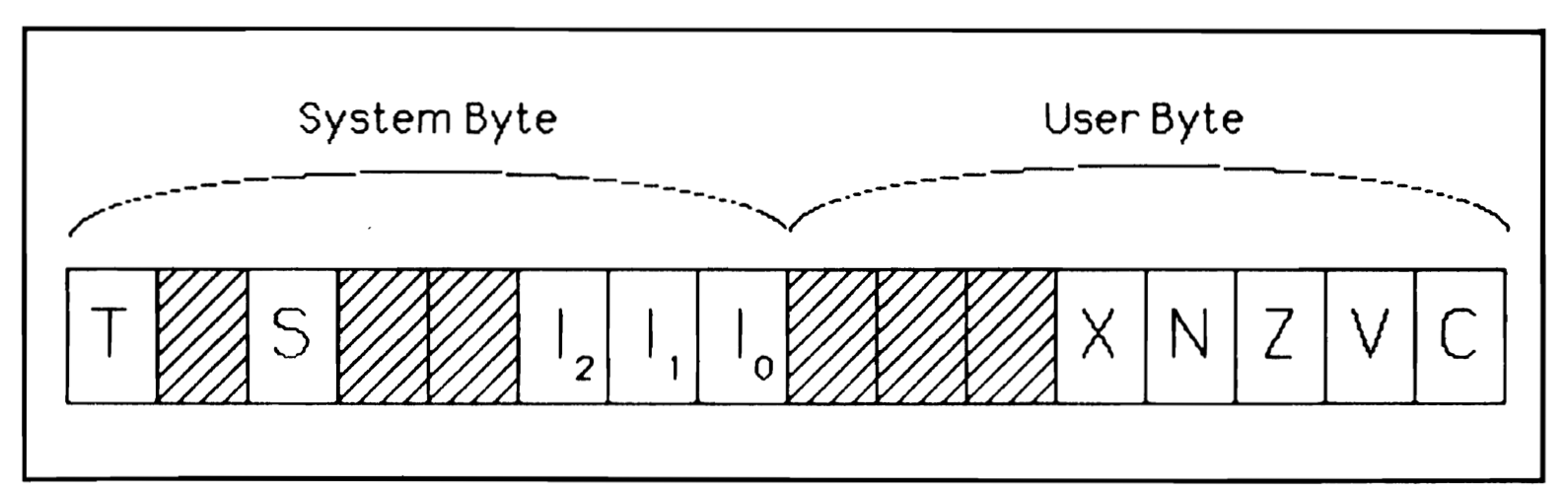
Wat er uit UAE komt, en ingebouwd is in onder andere Basilisk II, FS-UAE, Hatari, lijkt ook compleet ontoegankelijk voor eigen gebruik.

**Update 1 mei 2017:** Het is gelukt om Musashi te compilen. Stappenplan:

* Edit de file m68kconf.h:
  + Zet de mame optie op <off>
  + Zet drie opties voor processors 68010 en verder eventueel op off
  + Ergens achterin staat een optie voor alignment van words opcodes etc… op even adressen en het genereren van exceptions 🡪 dit is de normale modus van een standaard 68000
  + Laat de INLINE definitie voor wat deze is; of: maak het *static inline* (*static* is hier essentieel)
* gcc m68kmake.c 🡪 run daarna ./a.out
* m68kmake.c, a.out en m68k\_in.c kunnen nu worden gewist
* Definieer de in m68k.h genoemde functies (memory management), en de boel kan worden gelinkt.

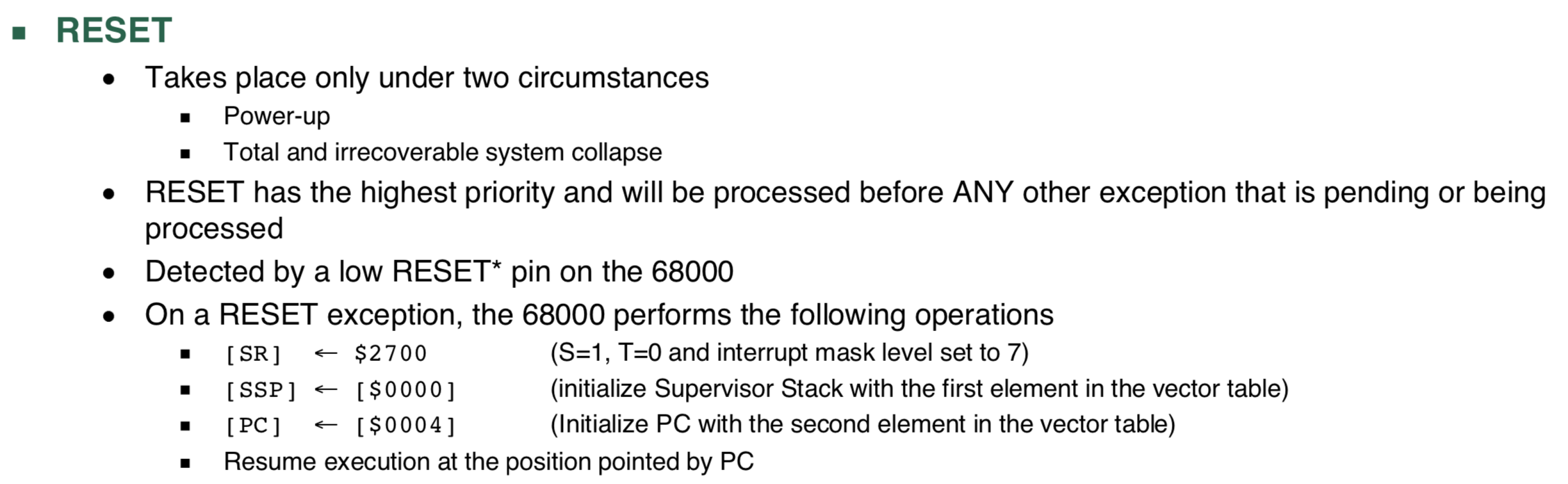
# Specifiek 68000

* Er zijn in de basis 23 address lines. Goed voor het adresseren van direct 2^23 adressen. Dat is de helft van 16mb (24 bits), maar hiermee worden eenheden van words geadresseerd (geen bytes). Dus toch 16mb. De consequentie is dat instructies en dataeenheden groter dan een byte altijd op een even adres moeten starten (zie ook de instelling hiervoor bij m68kconf.h).
* Het is wel mogelijk om individuele bytes te adresseren, hier worden twee andere lines voor gebruikt (naast die 23).
* De 68000 heeft zijn opcodes gecodeerd in words (16 bits). Er zit logica in de opbouw (bit patterns) van deze instructies (i.t.t. volgens mij bijv. De 65(c)02). Sommige opcodes vragen om een tweede opcode met de rest van de instructie.
* Hieronder de SR:



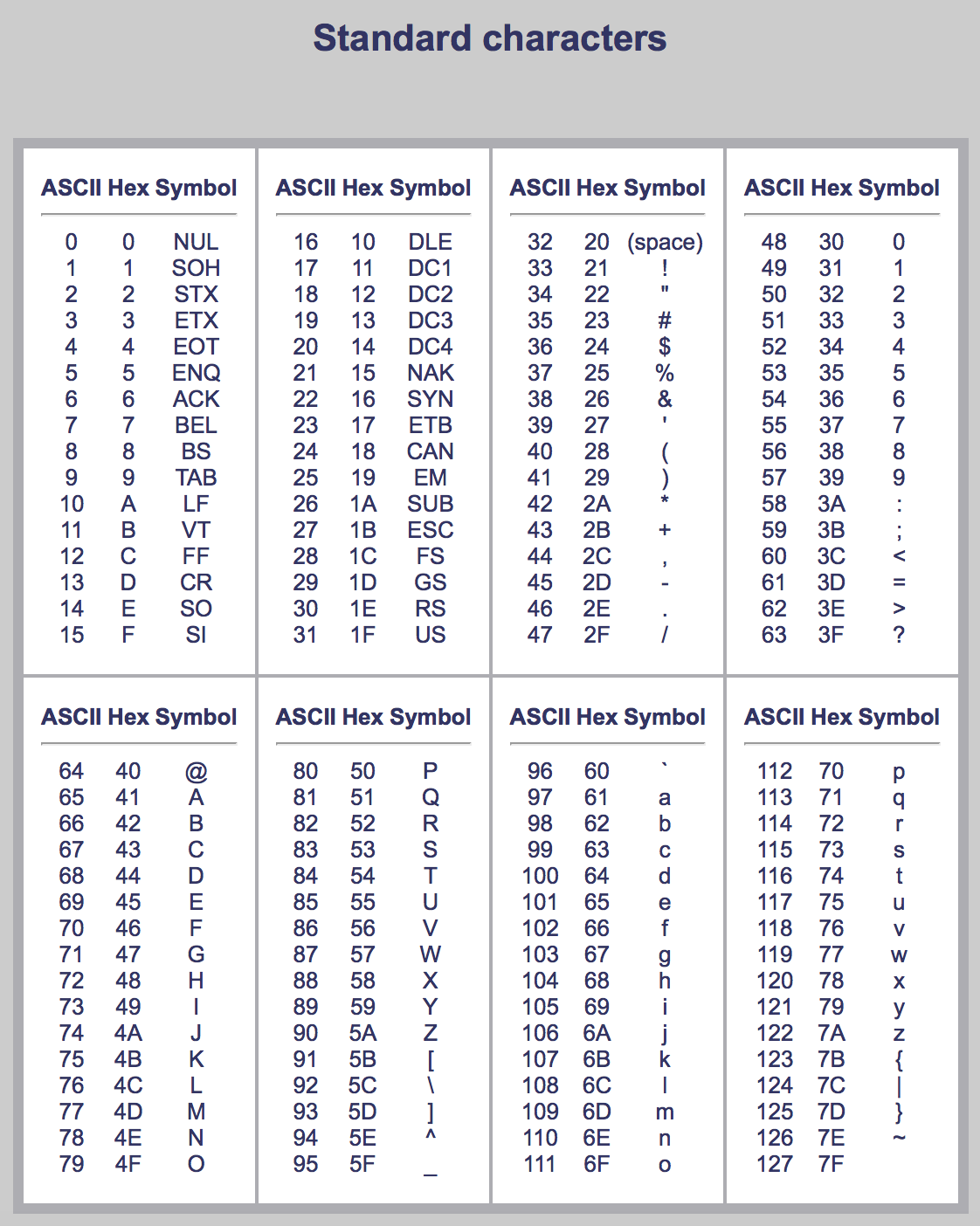
## Koude reset:

* Start met supervisor bit (=bit 13 in sr) aan. Hoe de cpu switcht van supervisor naar user mode en vice versa wordt goed uitgelegd op pagina’s 247-251 in Kelly (1985).
* Daarnaast informatie beschikbaar in exceptions pdf van course (hardware reset exception):



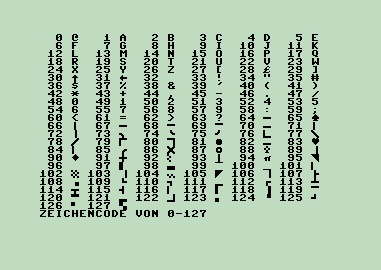
# ASCII to PETSCII conversion

## Standaard ASCII tabel



## PETSCII (C64) tabel

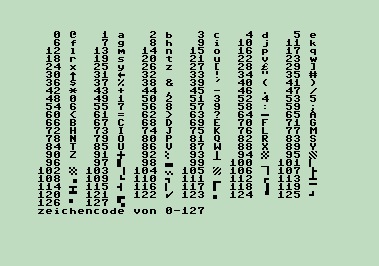
### Characters 0-127 charset 1



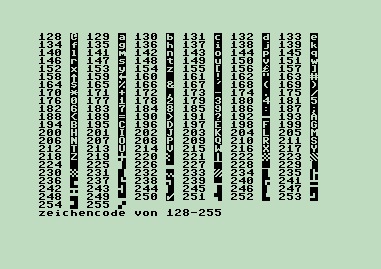
### Characters 128-255 charset 1



### Characters 0-127 charset 2



### Characters 128-255 charset 2



## Missende tekens uit standaard ascii

0x5c = \

0x5e = ^

0x5f = \_

0x60 = `

0x7b = {

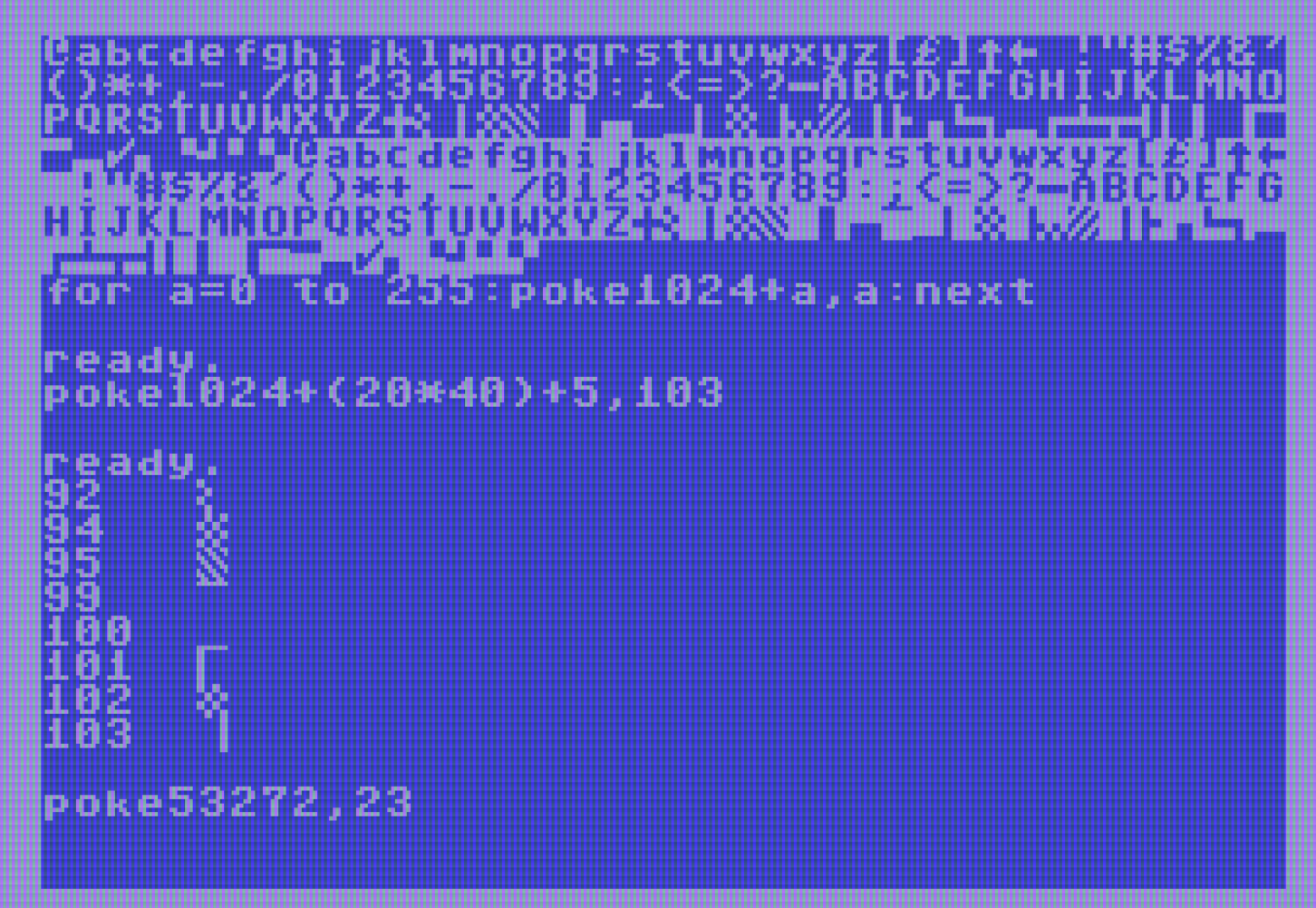
0x7c = |

0x7d = }

0x7e = ~

Welke tekens daarvoor opofferen? 92, 94, 95, 99, 100, 101, 102 en 103 dan maar.

Dat zijn 5c, 5e, 5f, 63, 64, 65, 66 en 67 in HEX.



# Calling Conventions, Stack operations and Function Calls

Principes:

* Nooit aannemen dat registers D0-D7 en A0-A5 nog hetzelfde bevatten nadat een functioncall heeft plaatsgevonden. Kortom: de caller is zelf verantwoordelijk voor het op de stack pushen en poppen van de waardes???
* Dan is de weg vrij om overal te kunnen rommelen met de registers
* Bij function calls parameters doorgeven via stack? Dus iets met *stackpointer* (A7) en *framepointers* (A6).
* Bij function calls return values teruggeven via de registers…
* Of toch helemaal anders?

# Opletten

* Movem instructie: ging op zijn bek omdat ik niet aangaf movem.l Hierdoor verdween bij het poppen van a0 het bovenste stuk van het register
* Per ongeluk bra gebruikt ipv bsr, tja, dan kom je nooit meer terug.
* Uitzoeken bra vs bra.s etcetera…

# Projectnamen

* C256
* C32
* Hitting the Virtual Metal