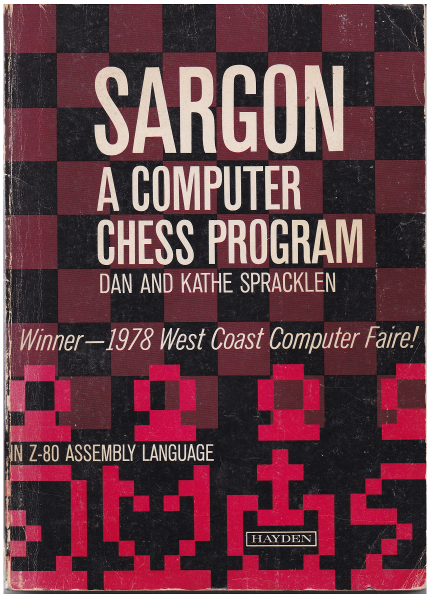


***Sargon von Akkad** (𒊬𒊒𒄀) war von 2356 bis 2300 v. Chr. König von Akkad. Mit Sargon von Akkad beginnt eine neue Ära in der Geschichte von Mesopotamien. Sargon und seine Gefährten verwendeten eine semitische Sprache und kamen aus westlichen Ländern. Seine Reichsgründung bedeutet insofern eine „Wasserscheide“ in Mesopotamiens Geschichte, als es das erste zentral verwaltete Großreich war, das über mehrere Generationen hin von derselben Herrscherfamilie regiert wurde.*
(Wikipedia)

Csargon

Chrilly Donninger, Mai 2021

Das Schach Programm SARGON von Dan und Kathe Spracklen genießt unter Computerschach-Interessierten einen ähnlichen Kultstatus wie der Steyr 15er unter Traktoristen. Es erschien 1978 und war um eine Klasse besser als die auf ähnlicher Hardware laufende Konkurrenz. Zieht man den Hardware-Faktor in Betracht, dann war es auch damaligen Supercomputer Programmen wie Chess und Cray-Blitz überlegen.
Das Programm lief ursprünglich auf dem Jupiter-III Mikrocomputer. Dieser war mit einem Z-80 8-Bit Mikroprozessor ausgestattet und hatte einen Bildschirm mit einer Auflösung von 96x128. Ich habe mich zu dieser Zeit noch mit Lochkarten herum geschlagen. Der Z-80 ist die erfolgreichste CPU und wird bis heute produziert. Er wurde auch in der DDR unter den Namen U880 nachgebaut. Der Z-80 war eine Meilenstein auf dem Weg zur modernen Informationsgesellschaft, SARGON war eine der ersten populären Anwendungen.
Die Spracklens haben den Kode in einem Buch veröffentlicht. Beim Aufräumen des Bücherkastens ist mir das Buch in die Hände gefallen. Daneben lag gleich die Z-80 Bibel Rodnay Zaks: Programming the Z80. Ich habe diese beiden Bücher vor einiger Zeit antiquarisch billig erworben. Diese Bücher sind inzwischen begehrte Sammlerobjekte. Das SARGON Buch notiert auf Amazon bei 62 Euro, der Z80 Klassiker bei 249. Wahrscheinlich sind es die einzigen Bücher in meiner sehr umfangreichen Sammlung, mit denen die Erben eine Freude haben werden (falls sie den Wert erkennen und es nicht wie der Rest im Altpapier-Container landet).
Ich dachte mir „*Jetzt oder nie*“ und entschied mich für das „*Jetzt*“. Das Ziel war SARGON so originalgetreu wie möglich mit C# nachzubauen. Der Nachbau wurde **CSargon** getauft (C wie C# oder auch wie Chrilly). Ich verwende C# im Moment auch für andere Projekte. C# ist ein bisserl langsamer als C. Das ist bei dieser Anwendung wurscht. Der Z-80 lief zwischen 2,5 und 4 MHz. Die schnellsten Befehle wie z.B. MOV A,B benötigen 4 Taktzyklen. Befehle wie DJNZ (Decrement and Jump if Not ZERO) 8 oder 13 Taktzyklen, ja nachdem ob der Sprung ausgeführt wird (13) oder nicht (8). Bei einem originalgetreuen Nachbau muss man sowieso Warteschleifen einbauen.
Unter einem originalgetreuen Nachbau kann man Verschiedenes verstehen. Der Z-80 hat in den ersten 3 Takten eines Befehls den ersten Byte-Opcode gelesen. Falls der Opcode wie bei MOV A,B nur 1 Byte lang war, hat er diesen im 4. Takt ausgeführt. Bei DJNZ muss zusätzlich der 1 Byte Sprung (man springt relativ zum aktuellen Programm Counter PC) in weiteren 3 Takten geladen und ein Decrement des B-Registers durchgeführt werden. Anschließend wird das bei dieser Operation gesetzte Zero-Flag überprüft und je nach Status der Sprung ausgeführt.
Bei einem genauen Nachbau des Z-80 würde auch der Simulator mit 2,5 MHz getaktet und das Einlesen des ersten Opcodes in 3. Schritte zerlegt. Wie bei den meisten Simulatoren wird bei CSargon die gesamte Operation in Einem ausgeführt. Es wird erst am Ende des Befehls eine Pause eingelegt.

Als Vorlage habe ich den C# Simulator von Marco Cecconi verwendet. Dieser hat jedoch in der CMP (Compare) Routine einen giftigen Bug. Bei der Befehlssequenz

```
MVI A, 2        ; Lade den Akkumulator mit der Brettreihe 2
CPI 9           ; Test ob der Zug über die 9-Reihe springt.
JRC ILLEGALROW  ; Wenn Carry gesetzt, springe zu „Illegaler Zug“
```

Der Cecconi Simulator setzt bei der Operation `CPI 9` nicht das Carry Flag. `CPI` führt die Operation A-Register – 9 aus. Das A-Register wird nicht verändert, es werden nur die Flags gesetzt. Der Cecconi Simulator setzt das Carry, wenn das Resultat größer gleich 256 ist. Tatsächlich muss es auch gesetzt werden, wenn das Resultat einer Subtraktion von 2 positiven Zahlen negativ ist. Das Problem trat glücklicher Weise bereits bei der Zugeingabe auf. Der Befehl wird in der Engine oft verwendet und wäre dort weit schwieriger zu finden gewesen.

SARGON wurde mit dem TdlAssembler geschrieben. Der Z-80 war aufwärts kompatibel zum 8080 von Intel. Es wurden nur einige Befehle hinzugefügt, die insbesondere die Adressierung des RAMs wesentlich erleichtern und beschleunigen. Der TdlAssembler ist in diesem Geiste geschrieben. Er verwendet die 8080 Mnemonics und nicht die des Z-80. Das SARGON Buch enthält einen Anhang mit Mnemonics Conversions. Das Buch von Rodnay Zaks verwendet nur die Zilog Schreibweise. Kathe Spracklen hat ein eigenes Z-80 Assembler Buch geschrieben, in dem beide Schreibwesen vorkommen. Vermutlich gibt es brauchbaren TdlAssembler Kode. Ich habe den Assembler jedoch gänzlich neu geschrieben. Der CSargon Assembler weicht in der Syntax geringfügig vom Original ab.
Statt `PAWN = 1` schreibt man `_EQU PAWN 1`.
`.BYTE`, `.WORD`, `.LOC` wurden durch `_BYTE`, `_WORD`, `_LOC` ersetzt.
Die Änderungen erleichtern das Parsen des Kodes. `PAWN` könnte auch ein Label sein. Die neue Schreibweise vermeidet ein Look-Ahead. Dasselbe gilt für `.BYTE`. „.“ wird auch als aktueller Wert des Programm Counter verwendet. Man springt im Kodebeispiel über den `DCR A` Befehl.

```
CPI   BPAWN       ; Is it a black Pawn ?
JRNZ  .+3         ; No-Skip
DCR   A           ; Decrement for black Pawns
ANI   7           ; Get piece type
```

Persönlich halte ich diese Schreibweise für schlechten Stil. Ich würde vor `ANI 7` ein Label `PCTYPE:` setzen und `JRNZ PCTYPE` schreiben. Es ist Aufgabe des Assemblers die Größe des Sprunges zu berechnen. Wahrscheinlich hat einst jeder Programmierer, der etwas auf sich hielt, es so wie die Spracklens geschrieben. Er dokumentierte damit, dass er wusste, wie lang der OpCode eines Befehls war (man muss auch die 2 Bytes des `JRNZ` Befehls dazu rechnen).
Von diesen minimalen rein syntaktischen Eingriffen abgesehen habe ich den Kode der Engine nicht angerührt. Man muss den SARGON Kode nicht abtippen. Man kann ihn auch von der Homepage von Andre Adrian herunter laden. Andre hat bei der Platzierung (`_LOC` Befehl) und bei der Größe des Move-Stacks Änderungen vor genommen. Ich habe es wieder auf die Originalwerte der Spracklens zurück gestellt. Das Herunterladen des Kodes war ein Fehler. Ich würde sowohl den Z-80 Assembler als auch den Programmkode weit besser verstehen, wenn ich mir die Mühe gemacht hätte, Zeile für Zeile, Seite für Seite, abzutippen. Man kann dabei Fehler einbauen. Allerdings ist die Gefahr nicht sehr groß, da einem meistens der Assembler mit „OpCode/Label not found“ auf die Finger klopfen würde.

Der Assembler verwendet zur Auflösung von Vorwärts Sprüngen zwei Durchgänge. Es wird kein HEX-File angelegt, sondern das RAM des Simulators direkt belegt. Der Assembler wird sogar bei jedem Newgame neu angeworfen. Dadurch wird auf einfache Art und Weise garantiert, dass man in einem sauberen Zustand beginnt. Der zusätzliche Aufwand ist auf einem modernen PC wurscht. Das Assemblieren dauert ca. 1 Sekunde.

```
;*********************************************************
; BOARD    --    Board Array. Used to hold the current position
;          of the board during play. The board itself
;          looks like:
;          FFFFFFFFFFFFFFFFFF
;          FFFFFFFFFFFFFFFFFF
;          FF0402030506030204FF
;          FF0101010101010101FF
;          FF0000000000000000FF
;          FF0000000000000000FF
;          FF0000000000000060FF
;          FF0000000000000000FF
;          FF8181818181818181FF
;          FF8482838586838284FF
;          FFFFFFFFFFFFFFFFFFFF
;          FFFFFFFFFFFFFFFFFFFF
;          The values of FF form the border of the
;          board, and are used to indicate when a piece
;          moves off the board. The individual bits of
;          the other bytes in the board array are as
;          follows:
;          Bit 7 -- Color of the piece
;               1 -- Black
;               0 -- White
;          Bit 6 -- Not used
;          Bit 5 -- Not used
;          Bit 4 --Castle flag for Kings only
;          Bit 3 -- Piece has moved flag
;          Bits 2-0 Piece type
;               1 -- Pawn
;               2 -- Knight
;               3 -- Bishop
;               4 -- Rook
;               5 -- Queen
;               6 -- King
;               7 -- Not used
;               0 -- Empty Square
;*********************************************************
_EQU BOARD  .-TBASE
BOARDA:     _BLKB 120
```

Die Engine verwendet ein konventionelles 12x10 Brett. Es gibt unten und oben ein Rand von 2 Reihen, links und rechts von einer Spalte. Der Springer springt quasi von h4 auf den linken Rand von a5 bzw. a3. Vermutlich wäre ein 8x16 vulgo 0x88 Brett effektiver. Es ist nur um 8 Byte größer, erleichtert jedoch die Abfrage, ob ein Zug das Brett verlassen hat. Die 0x88 Darstellung war 1978 noch nicht erfunden. Generell würde ich SARGON als damaligen State of the Art bezeichnen. Was es auszeichnet ist die handwerkliche Perfektion der Implementierung. Ein Beispiel dafür ist die Anweisung

```
_EQU BOARD  .-TBASE
```

Der Z-80 hat zwei Indexregister X und Y. Zum Registerwert kann man noch einen signed Byte-Offset hinzufügen. Die Tabellen sind so angeordnet, dass man diese Register möglichst wenig nachladen muss. Die Operation dauert 14 Takte. Der Intel 8080 hatte noch kein X und Y Register. Um kompatibel zu bleiben haben die Zilog Ingenieure die beiden noch freien Opcodes 0DDH (X) und 0FDH (Y) als Kennung für die Erweiterungen verwendet. Aus diesem Grund sind die Register-Ladebefehle um 1

Byte länger und 3 Takte langsamer. Das effektive Design der Tabellen war daher ein wichtiger Optimierungsschritt. Wenn man ein Programm in C schreibt, kann und braucht man sich um solche Details nicht mehr kümmern .

Ein einfaches Beispiel für die effektive Anordnung der Tabellen ist die Routine für das Aufstellen der Figuren in der Grundstellung. Das Brett wird im ersten Durchgang komplett mit dem Rand -1 bzw. 0FFH belegt. In der nächsten Schleife werden die Figuren Spaltenweise aufgestellt.

Anmerkung: Kommentare mit C.D. sind von mir eingefügt, alles andere ist Original.

Die Anweisung

```
MOV A,-8(X)
```

macht sich zu Nutze, dass im Memory direkt unterhalb des Brettes die Figurenpositionen definiert sind.

```
_EQU PIECES .-TBASE
      _BYTE 4,2,3,5,6,3,2,4
```

In der Spalte a wird zunächst der Turm (4) und die übrigen Figuren der a-Linie aufgestellt. Dann rückt man mit dem Befehl

```
INX X
```

eine Spalte weiter. Es kommt der Springer dran …

```
;*********************************************************
; BOARD SETUP ROUTINE
;*********************************************************
; FUNCTION: To initialize the board array, setting the
;           pieces in their initial positions for the
;           start of the game.
;
; CALLED BY:      DRIVER
;
; CALLS:    None
;
; ARGUMENTS:      None
;*********************************************************
INITBD: MVI B,120       ; Pre-fill board with -1's
      LXI   H,BOARDA
IB1:  MVI   M,-1
      INX   H
      DJNZ  .-3
      MVI   B,8           ; C.D. 8-Spalten von a bis h.
      LXI   X,BOARDA      ; C.D. Lade nur einmal das X-Index Register
IB2:  MOV   A,-8(X)           ; Fill non-border squares
      MOV   21(X),A           ; White pieces
      SET   7,A         ; Change to black
      MOV   91(X),A           ; Black pieces
      MVI   31(X),PAWN  ; White Pawns
      MVI   81(X),BPAWN ; Black Pawns
      MVI   41(X),0           ; Empty squares
      MVI   51(X),0
      MVI   61(X),0
      MVI   71(X),0
      INX   X                 ; C.D. Nächste Spalte.
      DJNZ  IB2               ; C.D. Mit Spalte h fertig?
      LXI   X,POSK            ; Init King/Queen position list
      MVI   0(X),25
      MVI   1(X),95
      MVI   2(X),24
      MVI   3(X),94
      RET
```

Ein netter Trick ist der Befehl

```
ANA   A     ; Clear carry flag
```

Die Operation ist A AND A. Es wird der Wert im Akkumulator A nicht verändert. Im ersten Moment denkt man sich: Hmmm, was soll das. Es steht die Auflösung des Rätsels als Kommentar dabei. Der Z-80 hat keinen Befehl zum Löschen des Carry-Flags. Warum auch: Es geht auch so (oder mit `ORA A`). Man könnte den Akkumulator mit dem Befehl

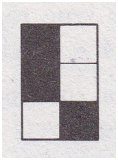
```
MVI A,0
```

auf Null setzen. Der Befehl ist 2 Bytes lang. 1 Byte für den Opcode, 1 Byte für den Operand 0. Er dauert 2x3 + 1 = 7 Takte. Der versierte Bitschnitzer schreibt hingegen

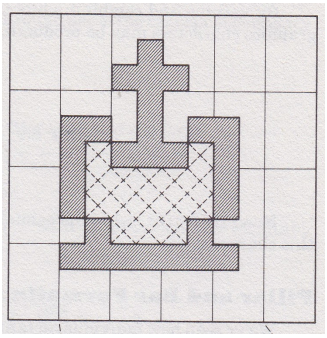
```
XRA   A     ; Zero
```

A XOR A ergibt auch null. Es ist nur 1 Byte lang und benötigt 4 Takte. Man hat 1 Byte und 3 Takte eingespart. Es ist in Zeiten von dicken, fetten Frameworks in denen „*Hello World*“ 200 MByte groß ist, erfrischend, wenn man solche Zeilen sieht.

Das GUI:



Der Bildschirm des Jupiter III war in 2x3 Rechtecke aufgeteilt. Jedes Rechteck entsprach einem Byte. Die 6 Pixel wurden durch Bit 0-5 gesetzt (Bit-7 war immer auf 1, Bit-6 wurde ignoriert). Das Bild entspricht dem Wert 0A6H. Ein Schachfeld bestand aus 12x12 Pixels bzw. 6x4 Rechtecken. Wobei der linke und rechte Rand aus der Feldfarbe besteht. Die Figur selbst ist eine 4x4 Byte-Matrix. Jeweils in vierfacher Ausführung. Weiß/Schwarz auf weißem /schwarzem Feld.



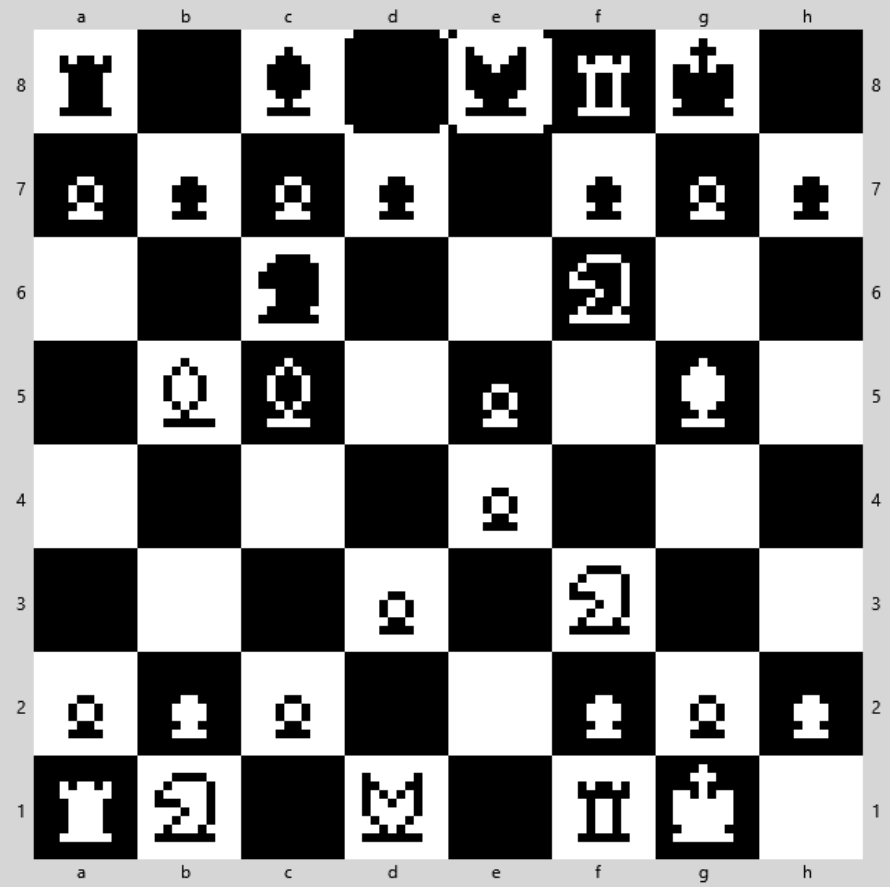
Das Bild zeigt das generelle Muster für den König. Die schraffierten Felder bilden den Kern der Figur. Er zeigt die Figurenfarbe an. Gemäß dem Motto „*möglichst originalgetreu*“ habe ich diese Darstellung übernommen. Das Schachbrett würde bei einer Darstellung von 96x96 Pixels wie eine Briefmarke aussehen. Jedes Pixel entspricht daher einem 6x6 Quadrat.
Ich habe mir noch die Freiheit genommen, das von- und nach-Feld des letzten Zuges an den 4 Ecken mit der gegenteiligen Feldfarbe zu markieren. Die Notation ist im Buch nicht definiert. Laut den Screenshots am Netz hat sie etwas anders als in CSargon ausgesehen.

SARGON ist in der untenstehenden Stellung 1 nach meinem Zug 6. Läufer c1-g5 mit 6 … Dame d8-e8 aus der Fesselung gezogen. Der Zug Dame d8-e8 zeigt eine Stärke und eine Schwäche des Programms. SARGON erkennt Fesselungen. Das ist für ein Programm das für die Engine 6 KByte zur Verfügung hat (2 KByte werden für IO und Grafik verwendet) nicht selbstverständlich. SARGON 1 hat keine Ruhesuche, sondern einen gefinkelten Static-Exchange Evaluator. Die Erkennung von Fesselungen ist Teil dieses Kodes. In späteren Versionen wurde der Exchange Evaluator durch eine Ruhesuche ersetzt.

Nach 7) g5xf6, g7xf6 hat Schwarz jedoch eine ziemlich kaputte Königsstellung. Das Partie war auch schnell zu Ende:
Chrilly v. CSargon:
1. e2-e4,e7-e5, 2. g1-f3,b8-c6 3. f1-b5,g8-f6 4. d2-d3,f8-c5 5. c1-g5,0-0 6. 0-0,d8-e8 7. g5xf6,g7xf6 8. f3-h4,a7-a6 9. b5-a4,b7-b5 10. a4-b3,c8-b7 11.d1-h5,g8-g7 12.h4-f5,g7-h8 13.h5-h6,e8-d8 14.h6-g7

Man muss zu SARGONs Ehrenrettung dazu sagen, dass ich gegen die 1 Ply Suchtiefe gespielt habe. Die Standardeinstellung ist Suchtiefe 2. Da wehrt sich das Programm schon wesentlich besser. Bei Suchtiefe 3 muss man schon viel Geduld haben. Theoretisch kann man bis zu Suchtiefe 6 einstellen.

	Human	CSargon
1.	e2-e4	e7-e5
2.	g1-f3	b8-c6
3.	f1-b5	g8-f6
4.	d2-d3	f8-c5
5.	0-0	0-0
6.	c1-g5	d8-e8
7.		

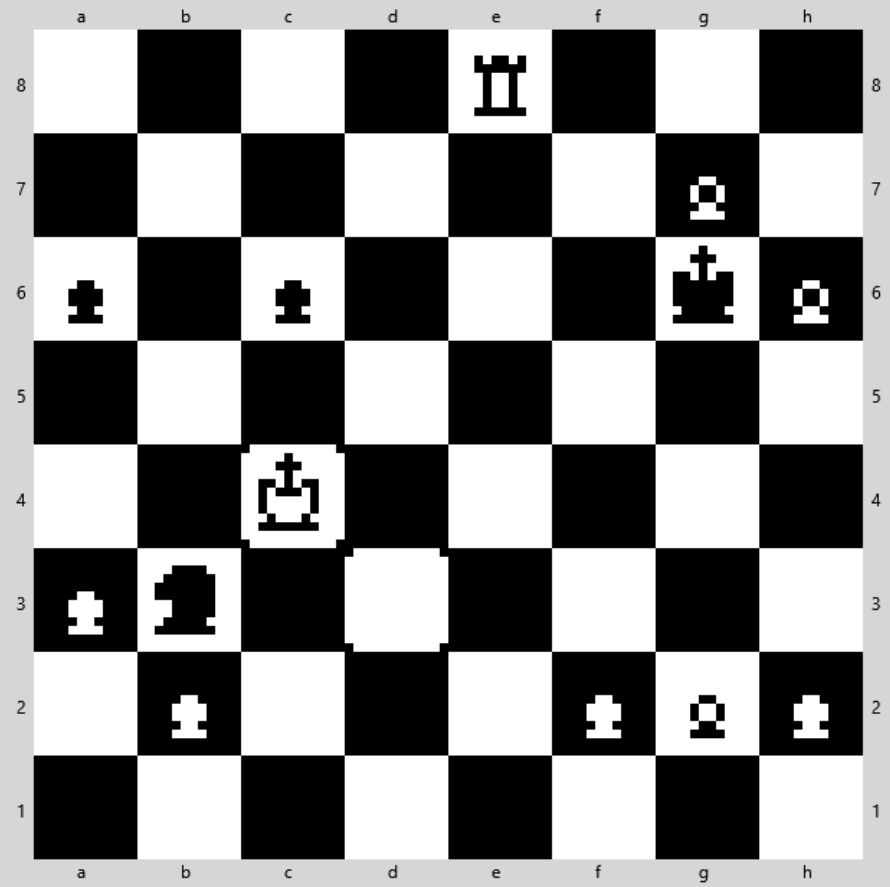


Stellung 1

In modernen Programmen wird die Stellungsbewertung angezeigt. Ich wollte auch bei CSargon die Bewertung ausgeben. Es gibt in der Engine eine Variable SCORE. Ich habe mir die Dame fressen lassen, der SCORE war 76. Im nächsten Zug waren es jedoch nur mehr 10. Soweit ich im Kode gesehen habe, wird der Score zu Beginn wieder auf 0 zurück gesetzt. Die Bewertung ist relativ zur Ausgangsstellung. Man hat beim Z-80 nur 8-Bit Arithmetik zur Verfügung. Wenn man den Wert eines Bauern auf 8 setzt und die positionellen Terme in 1/8 Bauern berechnet, läuft man schnell in einen Over/Underflow. Die Spracklens haben das Problem auf diese Weise elegant gelöst. Die absoluten Werte sind für die Zugauswahl egal. Es muss nur der beste Zug die meisten Punkte bekommen. Nachdem ich auch in den Screenshots keine Bewertung gesehen habe, habe ich es sein lassen.

Das Original verwendet zur Ein- und Ausgabe Betriebssystem Aufrufe des Jupiter III. Ich hatte keinen Ehrgeiz den Jupiter nachzubauen. Ich habe daher auf den Zilog Erweiterungstrick zurück gegriffen. Der Opcode 0FFH (soweit ich gesehen habe, ist es der einzig noch freie) definiert eine CSargon Erweiterung.
Die erweiterten Befehle sind `DSPLMV` (zeige Zug an), `RDMV` (Lese Zug ein), `DSPLBD` (Zeige Brett an), `DSPLMVNO` (Zeige Zugnummer an), `DSPLMSG` (Display Message Box), `DSPLWLC` (Display Welcome). Diese Befehle rufen quasi das C#-WPF „*Betriebssystem*“ auf.

	CSargon	Human
1.	e2-e4	e7-e5
2.	g1-f3	b8-c6
3.	d2-d4	d7-d6
4.	c1-g5	f8-e7
5.	g5xe7	g8xe7
6.	f1-b5	c8-d7
7.	b5xc6	d7xc6
8.	b1-c3	a7-a6
9.	d4xe5	d6xe5
10.	f3xe5	d8xd1
11.	a1xd1	a8-d8
12.	d1xd8	e8xd8
13.	e5xf7	d8-e8
14.	f7xh8	e8-f8
15.	0-0	f8-g8
16.	f1-d1	g8xh8
17.	d1-d8	e7-g8
18.	c3-d5	c6xd5
19.	e4xd5	c7-c5
20.	d5xc6eP	b7xc6
21.	g1-f1	h7-h6
22.	f1-e2	h8-h7
23.	d8-d6	g8-e7
24.	d6-d7	e7-d5
25.	d7-f7	h7-g6
26.	f7-f8	d5-b4
27.	a2-a3	b4xc2
28.	e2-d3	c2-a1
29.	f8-e8	a1-b3
30.	d3-c4	b3-d2
31.	c4-d3	d2-b3
32.	d3-c4	b3-d2
33.	c4-d3	d2-b3
34.	d3-c4	

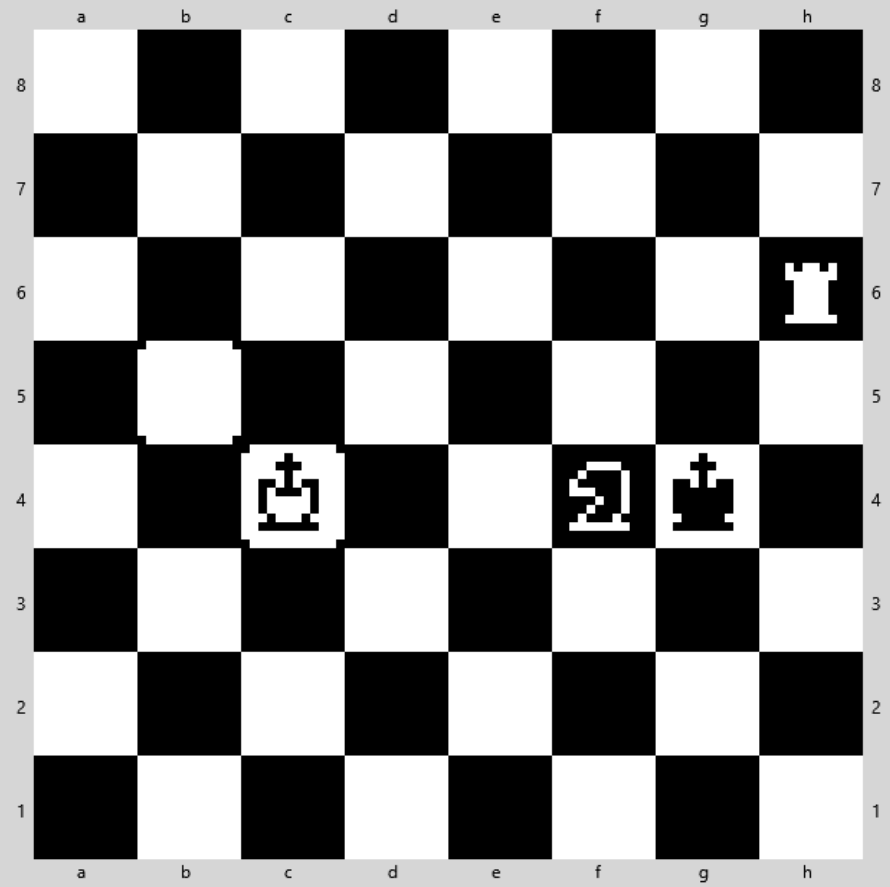


Stellung 2

Ich habe schon ewig nicht mehr Schach gespielt und war schon froh, dass ich noch die Regeln kann. Damit habe ich gegenüber SARGON schon einen gewissen Vorteil. In dieser Partie habe ich mit Schwarz eine Springergabel übersehen. Es kam zu dieser für SARGON klar gewonnen Stellung. SARGON zieht jedoch nach Springer b3-d2 König c4-d3 und wenn der Springer mit d2-b3 wieder zurück geht, greift er ihn wieder mit König d3-c4 an. Das Spiel lässt sich in alle Ewigkeit wiederholen, da SARGON keine Zugwiederholung erkennt. Nachdem ich dieses Feature ausgetestet habe, wollte ich auf ehrenhafte Weise Remis halten und habe in die Stellung 3 (unten) abgewickelt. Möglicher Weise würde ein Programm mit Tablebases nach ein paar Millisekunden so etwas wie Matt in 37 ausspucken. Ich spielte jedoch nicht gegen Gott sondern gegen SARGON und bin sicher, dass es mich nicht Matt setzen kann. Es kennt auch keine 50 Züge Regel.

Ich finde solche Fehler sympathisch. Gegen ein modernes Programm braucht man in der Stellung 2 nicht mehr weiter spielen. Man wird gnadenlos abgeräumt. Genau genommen sollte man bereits in der Grundstellung aufgeben um sich dieses Schicksal zu ersparen. Bei Verhandlungen des Hydra-Teams mit GMs ging es nie um die Sieges- sondern um die Antritts-Prämie. Wir nannten es intern die Abwatsch-Prämie oder auch Schmerzensgeld. Kein vernünftiger Mensch spielt heute noch zum Spaß gegen die stärksten Programme.

	CSargon	Human
41.	c4-d3	d2-b3
42.	d3-c4	b3-a5
43.	c4-b4	a5-b7
44.	e8-e7	b7-d8
45.	e7-d7	d8-f7
46.	b4-c5	f7-e5
47.	d7-d8	a6-a5
48.	b2-b4	a5xb4
49.	a3xb4	g6-f5
50.	b4-b5	c6xb5
51.	c5xb5	g7-g5
52.	d8-d6	e5-g4
53.	f2-f3	g4xh2
54.	d6xh6	h2-f1
55.	b5-c4	f1-e3
56.	c4-d3	e3xg2
57.	d3-c4	g5-g4
58.	f3xg4	f5xg4
59.	c4-b5	g2-f4
60.	b5-c4	

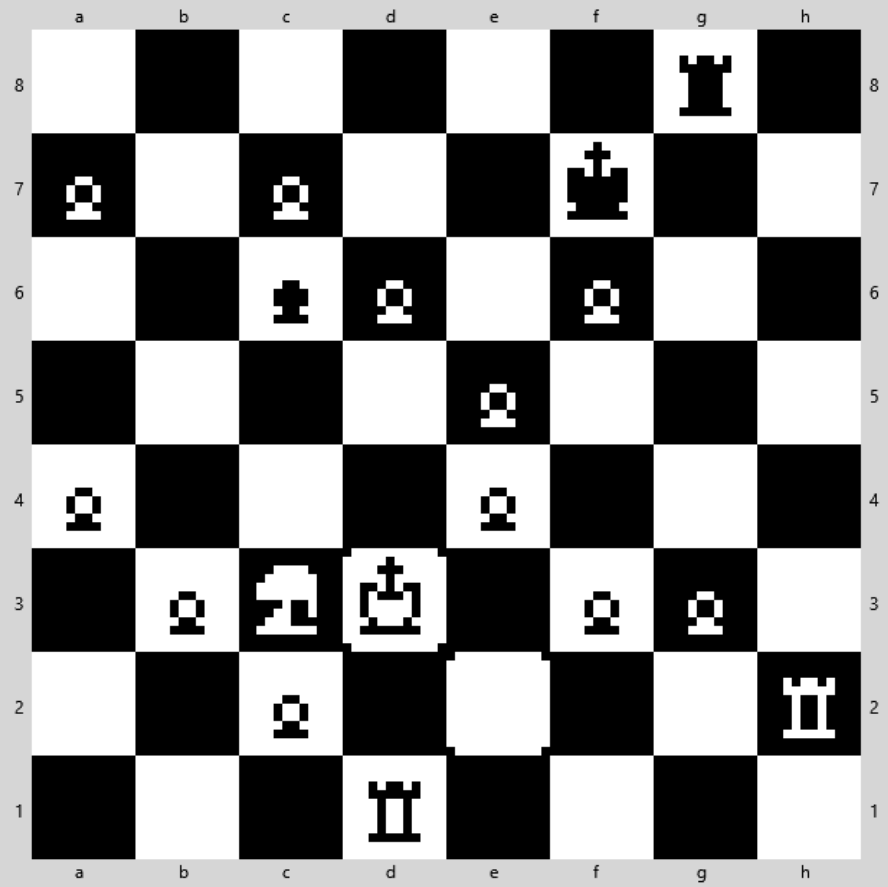


Stellung 3

Wenn man die Schwächen des Programms einmal erkannt hat, ist es selbst für einen Patzer wie dem Autor relativ leicht diese auszunützen. Wobei ich als langjähriger Computerschach Profi auch einen Blick für Programmblödheiten habe. Ich habe lange genug darunter gelitten. Die Dame aus der Fesselung weg bewegen und sich dafür die Königsstellung zu ruinieren ist ein derartiges Muster. Ich habe es auch in einer weiteren Partie mit Schwarz erfolgreich angewandt. SARGON spielt diesmal auf Suchtiefe 2 und verteidigt sich in Folge wesentlich besser. Es hat aber auch mein Königsangriff Verbesserungspotential.

CSargon, Chrilly
1. e2-e4,e7-e5 2. b1-c3,b8-c6 3. g1-f3,g8-f6 4. f1-c4,f8-c5 5. 0-0,d7-d6 6. c4-b5,c8-g4 7. b5xc6,b7xc6 8. d1-e1,g4xf3 9. g2xf3,f6-h5 10. g1-g2,h5-f4 11. g2-h1,d8-h4 12. a2-a4,f4-h3 13. d2-d4,c5xd4 14. c1-d2,h3xf2 15. h1-g2,h4-h3 16. g2-g1,f2-g4 17. f1-f2,g4xf2 18. d2-e3,d4xe3 19. e1xe3,h3-h4 20. e3xf2,h4xf2 21. g1xf2,a8-b8 22. b2-b3,g7-g5 23. a1-g1,f7-f6 24. g1-d1,e8-f7 25. f2-e3,h7-h5 26. h2-h3,b8-g8 27. d1-g1,h8-h7 28. g1-d1,g5-g4 29. h3xg4,h5xg4 30. e3-f2,g4-g3 31. f2-e2,h7-h2 32. e2-d3 (Stellung 4), g3-g2 33. d1-g1,h2-h1 34. c3-e2,h1xg1 35. e2xg1,g8-h8 36. g1-e2,h8-h1 37. c2-c3,h1-e1 38. d3-d2,e1xe2 39. d2xe2,g2-g1Q 40. c3-c4,g1-b1 41. a4-a5,b1xb3 42. e2-f2,b3xc4 43. f2-g3,f7-g6 44. g3-f2,c4-d3 45. f2-g3,g6-g5 46. g3-f2,g5-f4 47. f2-e1,d3-c2 48. e1-f1,f4xf3 49. f1-g1,c2-g2

	CSargon	Human
1.	e2-e4	e7-e5
2.	b1-c3	b8-c6
3.	g1-f3	g8-f6
4.	f1-c4	f8-c5
5.	0-0	d7-d6
6.	c4-b5	c8-g4
7.	b5xc6	b7xc6
8.	d1-e1	g4xf3
9.	g2xf3	f6-h5
10.	g1-g2	h5-f4
11.	g2-h1	d8-h4
12.	a2-a4	f4-h3
13.	d2-d4	c5xd4
14.	c1-d2	h3xf2
15.	h1-g2	h4-h3
16.	g2-g1	f2-g4
17.	f1-f2	g4xf2
18.	d2-e3	d4xe3
19.	e1xe3	h3-h4
20.	e3xf2	h4xf2
21.	g1xf2	a8-b8
22.	b2-b3	g7-g5
23.	a1-g1	f7-f6
24.	g1-d1	e8-f7
25.	f2-e3	h7-h5
26.	h2-h3	b8-g8
27.	d1-g1	h8-h7
28.	g1-d1	g5-g4
29.	h3xg4	h5xg4
30.	e3-f2	g4-g3
31.	f2-e2	h7-h2
32.	e2-d3	



Stellung 4

Das Eröffnungsbuch:
SARGON hat auch ein Eröffnungsbuch. Es spielt mit Weiß zufällig e4 oder d4, je nachdem ob im Interrupt-Counter Register Bit 0 gesetzt ist oder nicht. Bei der Simulation habe ich etwas geschummelt. Es wird das Bit-0 der aktuellen Zeit als Zufallsgenerator verwendet. Mit Schwarz spielt SARGON in Abhängigkeit vom 1. Zug des Gegners e5 oder d5. Das war‘s dann auch schon. Das Programm entwickelt sich in Folge vernünftig. Das „*die Dame zieht aus der Fesselung*“ Problem kann man mit einem Buch schwer flicken. Der menschliche Gegner kann mit einem unkonventionellen Zug das Programm aus dem Buch werfen und dann die Blödheit ausnützen.

Verfügbarkeit:
Wenn jemand am Kode (Visual Studio 2019 Projekt) und/oder am Programm interessiert ist, möge er mir unter c.donninger@wavenet.at eine Mail schreiben.

SARGON 1++ ???
Ich habe mir kurz überlegt das Programm zu verbessern. Eine naheliegende Idee ist den Static-Exchange Evaluator durch eine Ruhesuche zu ersetzen. Es passt aber alles zusammen. Man kann nicht einfach ein Element heraus nehmen und es durch ein Neues ersetzen. Man müsste alles neu schreiben. Die Spracklens haben das bei SARGON 2.5 auch gemacht. Davon abgesehen baut man in einem 15er Traktor auch keinen Turbolader ein. Ich habe das Ziel, SARGON möglichst originalgetreu nachzubauen, erreicht und sollte es damit gut sein lassen.

Mephisto I oder das Brikett 2.0!

Der Mephisto I von Thomas Nitsche erschien 1980. Er spielte in Europa und insbesondere in Deutschland eine ähnliche Rolle wie SARGON in den USA. Sein Spitzname war das „*Brikett*“. Ein Brikett nachzubauen wäre eine reizvolle Aufgabe. Als 8-Bit Mikroprozessor bietet sich der Atmel AVR an. Es sind u.A. die meisten Arduino Platinen mit dem AVR bestückt. Die Architektur ist von Haus aus für C-Compiler ausgelegt. Man kann das mit Assembler-Programmierung vermutlich nicht großartig verbessern. Darum geht es auch nicht.

Die beliebteste Plattform ist der Arduino Uno. Der vom Uno verwendete ATmega328P hat 2KByte RAM. Die 2 KByte kann man wegen der Harvard-Architektur des AVR rein für Datenstrukturen verwenden. 2 KByte sind trotzdem für ein starkes Programm Arschknapp (ich spreche aus leidvoller Erfahrung beim Projekt Tiger Chess für den Hitachi SuperH. Der SuperH hatte 4 KB). Der Arduino Mega 2560 ist mit dem ATmega 2560 bestückt. Dieser hat 8KByte RAM. Das sollte reichen. Das Problem bei diesem Projekt ist: Ich könnte mit einiger Sicherheit ein starkes Assembler-Programm schreiben. Ich kann jedoch mit absoluter Sicherheit kein Brikett bauen. Dazu bräuchte ich als Partner einen geschickten Hardware-Enthusiasten. Wenn der geneigte Leser so einer ist (und die Idee gut findet) oder so jemanden kennt, möge er sich bitte melden.

Es gibt mit dem Shah bereits so ein Brikett. Dieses hat den schwächeren 2 KByte RAM Prozessor eingebaut. Beim Design stand wohl mehr der Hardware-Maker Aspekt und nicht die Stärke des Schachprogramms im Vordergrund. CShah soll jedoch – für ein 8-Bit Mikroprogramm - möglichst stark spielen. „zu stark“ geht bei dieser Hardware eh noch nicht.

Es gibt Arduino Platinen mit ARM-Prozessoren. Ein C-Programm für den ARM zu schreiben finde ich witzlos. Das habe ich mit Nimzo 1-4 schon hinreichend gemacht.

Hydra-Light?

Eine Alternative ist ein FPGA-Entwicklungsboard. Es gibt bereits für 100 Euro das nette Terasic DE2-115 Nano. Man könnte auf dieser FPGA als Einstieg ebenfalls einen AVR-Core implementieren und dafür ein Programm schreiben. Das Programm sollte auf der FPGA und auf einem Brikett laufen. Im nächsten Schritt könnte man den Instruktionssatz für Schach optimieren. Z.B. unnötige Befehle wie die Multiplikation weg lassen, aber vor allem „*Superbefehle*“ wie z.B. „*erzeuge alle angegriffenen Felder*“ hinzufügen. Das würde sich in Richtung Hydra bewegen. Allerdings habe ich keinen Ehrgeiz Hydra 2.0 zu bauen. Es geht nur darum meine FPGA Kenntnisse aufzufrischen.

Verwendete Literatur:

Dan und Kathe Spracklen: SARGON – A Computer Chess Programm. Hayden 1978
Kathe Spracklen: Z-80 and 8080 Assembly Language Programming, Hayden 1979
Rodnay Zaks: Programming the Z80, 3rd Edition. Sybex 1982.
Robert Dunne: Computer Architecture Tutorial using an FPGA. Eigenverlag, 2020
Muhammad Ali Mazidi et al.: The AVR Microcontroller and Embedded Systems using Assembly and C. 2nd Edition, Eigenverlag, 2017.
Kathe and Dan Spracklen: First Steps in Computer Chess Programming, BYTE Oct. 1978