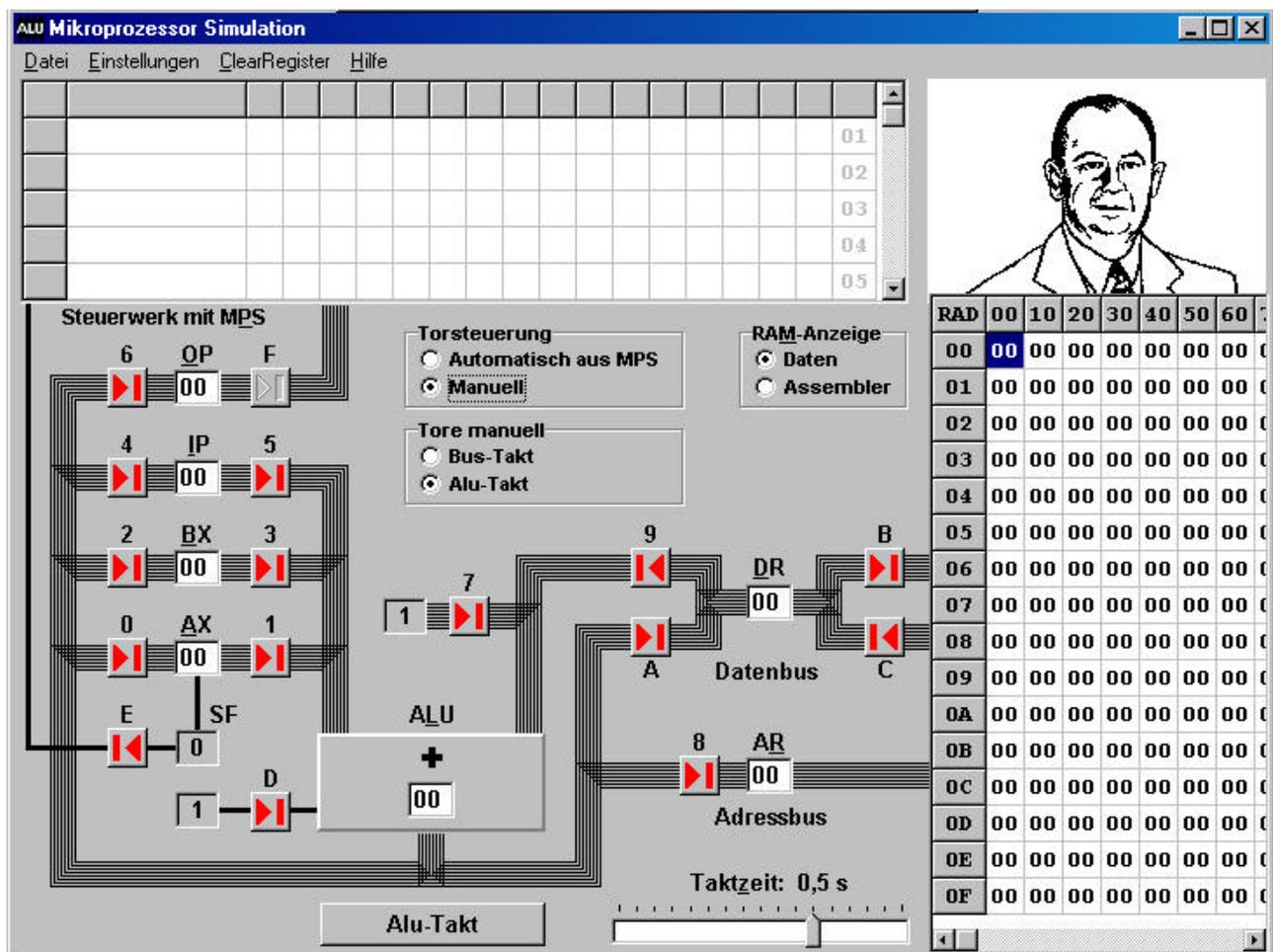


0 Allgemeines zu MIKROSIM

MIKROSIM ist ein Simulator, der alle wichtigen Befehlsarten, wie sie von realen Prozessoren (z.B. 8086, 80286, 80386 usw.) her bekannt sind, abarbeiten kann. Nachdem bei der Arbeit mit einer Assemblersprache klar wurde, was die einzelnen Befehle bewirken, verdeutlicht MIKROSIM, wie der Prozessor eines Computers solche Befehle und ganze Programme im einzelnen ausführt.

MIKROSIM besitzt einen Arbeitsspeicher (RAM) mit 256 Speicherplätzen, die von 00 bis FF (hexadezimal) nummeriert sind (hier werden nur die Speicherplätze 00-1F benutzt). Außerdem sind 6 Register (DR, AR, AX, BX, OP, IP), eine ALU (=arithmetisch logische Einheit) und ein Flag (SF) vorhanden. Die Register und die RAM-Plätze fassen jeweils ein Datenwort von 1 Byte (= 8 Bit). Bei der Ausführung eines Befehls müssen Daten von einer dieser Funktionseinheiten zur anderen transportiert werden. Um alle 8 Bit eines Datenwortes gleichzeitig übertragen zu können, sind dazu Bündel von je 8 Einzelleitungen notwendig, die Busse genannt werden. Bei MIKROSIM werden sie als Striche auf dem Bildschirm dargestellt. In diese Busse sind Tore eingebaut, die (zunächst von Hand, später auch automatisch) geöffnet und geschlossen werden können, so dass gezielte Datentransporte möglich werden.

Start der Simulation: MIKROSIM durch Doppelklick aufrufen
 Ende der Simulation: Datei / Ende oder [ALT]-F4
 Hilfe: Hilfe/Hilfe oder F1



1a Rechen- und Transportoperationen zwischen Registern

Register (DR, AR, AX, BX, OP, IP):

Die Namen der Register werden erst im Lauf der Übungen klar.

Die Inhalte einzelner Register können mit Hilfe von Maus und Tastatur direkt geändert werden. (ALT A geht zu Register AX).

Tore (von 0 bis F durchnummeriert):

Anklicken (linke Maustaste) eines Tores 0..E (F spielt eine Sonderrolle) wählt das entsprechende Tor zum Öffnen aus. Erneutes Anklicken macht die Auswahl wieder rückgängig. Die Tore können auch über die Tastatur mit STRG 0 bis STRG F ausgewählt werden.

Ausgangstore (ungerade Nummern) führen aus einem Register hinaus, Eingangstore (gerade Nummern) führen in ein Register hinein. Es wäre sinnlos und würde in der Realität zu einem Kurzschluss führen, wenn man mehr als ein Ausgangstor gleichzeitig zum selben Bus hin öffnen würde. MIKROSIM lässt eine solche Fehlbedienung nicht zu. Dagegen dürfen am gleichen Bus beliebig viele Eingangstore gleichzeitig geöffnet sein. Ist kein einziges Ausgangstor eines Busses geöffnet, so überträgt der Bus den Wert Null.

Die Tore können manuell oder automatisch angesteuert werden:

Takt:

Beim realen Prozessor sorgt ein Taktgenerator für den synchronisierten Ablauf aller Vorgänge. Bei MIKROSIM wird ein Takt mit Hilfe der Leertaste ausgelöst. Jeder Takt umfasst zwei Phasen. In der ersten öffnen die ausgewählten Ausgangstore. Sie werden wieder geschlossen, bevor in der zweiten Phase die ausgewählten Eingangstore öffnen.

Die Taktzeit wird durch einen Timer geliefert.

- Bus-Takt := Taktzeit; wechselnd werden Ausgangstore und Eingangstore geöffnet.

- ALU-Takt := 2*Taktzeit; es werden immer beide Bustakte durchgeführt.

Für einen großen Bereich von 0,01 s bis 10 s hat der Regler eine logarithmische Skala.

- „schnell“ arbeitet ohne Timer mit allen Anzeigen,

- „schneller“ arbeitet ohne Bus- und Tor-Anzeige,

- „sehr schnell“ ohne Cursor in MPS und RAM und

- „zu schnell“ ohne Register-Anzeige. Die schnelleren Modi sind für Schleifen interessant.

ALU (arithmetisch-logische Einheit):

Sie besitzt zwei Dateneingänge, und einen Ausgang für das Verknüpfungsergebnis. Dem Ausgang ist ein von außen nicht zugänglicher Zwischenspeicher (Latch) vorgeschaltet, der das Verknüpfungsergebnis für die kurze Zeit sichert, in der die Ausgangstore schon geschlossen, die Eingangstore aber noch nicht geöffnet sind. Über den Steuereingang lässt sich die von der ALU bewirkte Verknüpfungsoption festlegen (MIKROSIM: addieren/subtrahieren).

Übung 1a: (das Registerfenster darf nur bei a) benutzt werden!)

- a) DR := 09
- b) AX := DR, IP := DR
- c) BX := IP + DR
- d) IP := -1
- e) IP := IP + a
- f) AX := AX - BX
- g) setzen Sie alle Register auf 0

Frage: Wie viele Steuerleitungen braucht eine ALU, die 32 Operationen ausführen soll?

1b Speicherzugriffe

Der Speicher (RAM) ist über den Datenbus und den Adressbus mit dem Mikroprozessor verbunden. Der Datenverkehr wird ausschließlich über das Datenregister DR abgewickelt. Das Adressregister AR enthält die Quell- bzw. Zieladresse der Daten. Zur bequemen Eingabe besitzt MIKROSIM ein RAM-Fenster, das durch direkte Eingabe und über das Dateimenü gefüllt werden kann.

Übung 1b: (Fortsetzung)

- h) Füllen Sie die ersten drei RAM-Plätze mit den Zahlen 1, 3 und 5, ohne das RAM-Fenster direkt zu benutzen.
- i) Die Summe dieser Zahlen soll nach RAM[3], und zwar ohne Kopfrechnen!
- j) Löschen Sie das RAM (F9), bringen Sie dann den Wert 5 an die Adresse 1D und den Wert 3 an die Adresse 1E. Speichern Sie den neuen RAM-Inhalt unter dem Namen UEB1 in Ihr Homeverzeichnis, löschen Sie dann das RAM wieder.

2 Realisierung einzelner Maschinenbefehle durch Handsteuerung

In der Realität steht das Maschinenprogramm und damit auch jeder Operand im RAM und kann dem Mikroprozessor nur über DR zugeführt werden. Voraussetzung der folgenden Übungen ist es deshalb, dass der Operand eines jeden Maschinenbefehls vor dessen Ausführung (manuell) nach DR gebracht wird.

Übung 2:

Mit Hilfe von F3 soll die Speicherbelegung UEB1 geladen werden. Anschließend sollen die folgenden Maschinenbefehle von Hand ausgeführt werden. Kreuzen Sie im folgenden Schema jeweils an, welche Tore dazu der Reihe nach zu öffnen sind. Die kursiv geschriebenen Operanden sind jeweils vor Ausführung der Befehle von Hand nach DR zu bringen! Achten Sie auf die Bedeutung der eckigen Klammern.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E
a) mov AX, [1D]															
b) sub AX, [1E]															
c) mov [1D], AX															
d) inc [1F] (der Inhalt von AX darf dabei nicht verändert werden)															

Alle Vorgänge, die während eines Taktes (2 Phasen!) im Mikroprozessor ablaufen, werden durch die Angabe derjenigen Tore festgelegt, die dabei zu öffnen sind. Diese Angabe heißt Mikrobefehl. Jede Zeile des obigen Schemas entspricht also einem Mikrobefehl. Jeder Maschinenbefehl wird durch einen oder mehrere Mikrobefehle realisiert.

Frage: Wie könnte die Ausführung einer Sequenz von Mikrobefehlen automatisiert werden?

3 Automatische Ausführung eines einzelnen Maschinenbefehls

Neben dem Arbeitsspeicher (RAM) steht in MIKROSIM auch ein Mikroprogrammspeicher (MPS) zur Verfügung, in dem Sequenzen von Mikrobefehlen, also ganze Mikroprogramme abgelegt werden können. Nun kann das Steuerwerk selbst die in den einzelnen Mikrobefehlen angegebenen Tore zum Öffnen auswählen, ein Eingriff von Hand erübrigt sich. Vom Steuerwerk aus gehen hierzu Steuerleitungen zu jedem Tor. Diese Leitungen werden bei MIKROSIM der Übersichtlichkeit wegen nicht dargestellt.

Um Mikrobefehle einzugeben, klicken Sie mit der Maus in den entsprechenden Bereich auf dem Bildschirm. Ein Doppelklick (oder rechter Mausklick) setzt eine Markierung für das entsprechende Tor. Unter Einstellungen können Sie wählen, ob die Tornummer oder nur ein Kreuz angezeigt werden soll. Die Plätze des Mikroprogrammspeichers sind von 00 bis FF durchnummeriert, eine automatische Ausführung beginnt immer mit dem Befehl auf Platz 00. Einzelne Mikrobefehle können mit Strg Entf gelöscht werden, Platz für neue Zeilen schafft man mit Strg Einfg. Die Folgeadressen (s.u.) werden dabei automatisch angepasst.

Ein Mikrobefehl besitzt bei MIKROSIM 24 Bit. Die ersten 16 Bit sind von 0 bis F nummeriert und den gleichnamigen Toren zugeordnet (Steuerleitungen!). Sie lassen sich einzeln per Doppelklick oder durch Antippen der Leertaste manipulieren. Die letzten 8 Bit sind zu einer zweistelligen Hex-Zahl zusammengefasst und legen fest, bei welcher Platznummer die Ausführung des Mikroprogramms fortgesetzt wird (FA = Folgeadresse). Normalerweise ist FA um 1 größer als die aktuelle Platznummer (Voreinstellung), doch sind Abweichungen davon jederzeit möglich. So ist z.B. beim letzten Mikrobefehl einer zusammengehörenden Sequenz FA auf 00 zu setzen, weil dies für MIKROSIM das Signal ist, die automatische Ausführung anzuhalten. Links vor der Platznummer eines jeden Mikrobefehls ist Platz für eine kurze Bemerkung. Vor dem ersten Befehl einer Sequenz trägt man hier ein Mnemonic für den zu realisierenden Maschinenbefehl ein.

Übung 3a: (nn steht für ein Byte, das sich im Register DR befindet)

- Führen Sie den Assemblerbefehl INC [nn] manuell aus und notieren Sie das hierzu nötige Mikroprogramm (vgl. Übung 2e).
- Schreiben Sie das notierte Mikroprogramm in den MPS ab Platz 00. Schreiben Sie dabei "INC [nn]" als Bemerkung vor den Befehl auf Platz 00. Ändern Sie beim letzten Mikrobefehl FA auf 00 ab!
- Löschen Sie alle Register und das RAM manuell, belegen Sie DR mit 05, RAM[05] mit 03.
- Schalten Sie auf automatischen Betrieb um und lösen Sie einen Takt aus. Lösen Sie erneut einen Takt aus, sobald die Torsteuerung anhält.
- Experimentieren Sie auch mit verschiedenen Geschwindigkeiten.

Übung 3b: Löschen Sie den MPS durch Laden der Datei LEER.

- Speichern Sie die in Übung 2a, 2b, 2c bzw. 2d erstellten Mikroprogramme in den MPS so ab, dass die Sequenzen für die einzelnen Befehle auf den Plätzen 10, 23, 13, bzw. 30 beginnen. Schreiben Sie zu Beginn jeder Sequenz das entsprechende Mnemonic als Bemerkung und setzen Sie am Ende der Sequenz FA = 00.
- Sichere den Inhalt des MPS unter dem Namen UEB3 in Ihr Homeverzeichnis.

Frage:

Wie kann man erreichen, dass bei Antippen der Leertaste nicht das erste beste, sondern z.B. das auf Platz 30 beginnende Mikroprogrammstück automatisch ausgeführt wird?

4 Halbautomatische Ausführung eines Maschinenprogramms

In Abschnitt 3 wurde erläutert, dass die letzten 8 Bit eines Mikrobefehls die Folgeadresse FA bei der automatischen Ausführung bedeuten. Dies stimmt nur, wenn das Tor F (wie bisher immer) geschlossen ist. In Wirklichkeit wird die Folgeadresse nämlich durch ein nicht dargestelltes Addierwerk als Summe von FA und dem durch Tor F gelieferten Wert berechnet. Ist Tor F geschlossen, so liefert es den Wert 00, ist es geöffnet, so liefert es den Inhalt von OP. Tor F kann nicht von Hand, sondern nur per Mikrobefehl geöffnet werden.

Übung 4:

- Laden Sie UEB1 ins RAM und UEB3 in den MPS.
- Schreiben Sie auf Platz 00 des MPS einen Mikrobefehl, der bewirkt, dass nicht bei Platz 01 weitergemacht wird, sondern dass der Inhalt von OP darüber entscheidet, wo das Mikroprogramm fortgesetzt wird.
- Das in Übung 2 entwickelte Maschinenprogramm kann nun Befehl für Befehl (halbautomatisch) ausgeführt werden, indem (Registerfenster, F9!) in OP jedes Mal der Startplatz des Mikroprogramms für denjenigen Befehl eingetragen wird, der als nächster auszuführen ist. Außerdem muss der Operand für diesen Befehl jeweils in DR bereitgestellt werden:

Assemblerbefehl	Startadresse in OP	Operand in DR
mov AX,[1D]	10	1D
sub AX,[1E]	23	1E
mov [1D],AX	13	1D
inc [1F]	30	1F

Frage:

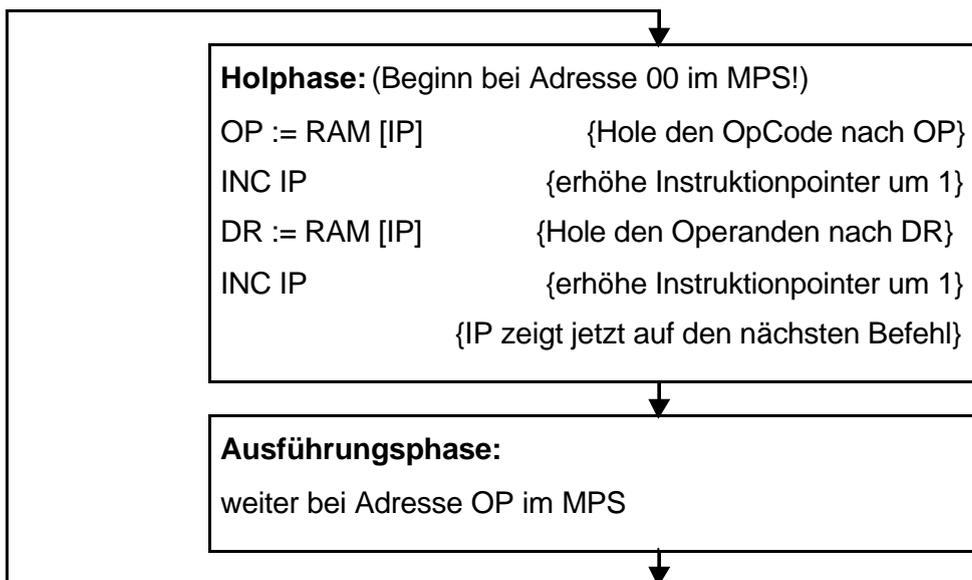
Was fehlt noch zur vollautomatischen Ausführung eines Maschinenprogramms?

5 Vollautomatische Ausführung eines linearen Maschinenprogramms

Damit ein Maschinenprogramm vollautomatisch ausgeführt werden kann, müssen alle seine Befehle komplett im RAM gespeichert sein (bisher standen dort nur die Daten). Hierzu ist es notwendig, für jeden Maschinenbefehl sowohl die Operation als auch den Operanden (falls vorhanden) durch eine zweistellige Hex-Zahl zu codieren. Als Operationscode bietet sich die Startadresse des zugehörigen Mikroprogramms im MPS an, der Operand ist ohnehin eine Zahl (je nach Adressierungsart verschieden interpretiert).

Im Normalfall stellen wir jeden Maschinenbefehl im RAM also durch zwei aufeinanderfolgende Bytes dar. Bevor ein Maschinenbefehl ausgeführt werden kann, muss das erste dieser beiden Bytes nach OP, das zweite nach DR gebracht werden (Holphase). Anschließend kann dann die Ausführung der für diesen Befehl zuständigen Mikroprogramme angestoßen werden (Ausführungsphase). Ist die Ausführung beendet, so muss sich die Holphase für den nächsten Maschinenbefehl anschließen. Damit der Mikroprozessor ihn findet, muss die RAM-Adresse dieses Befehls in einem Register zur Verfügung stehen. Wir benutzen IP als Befehlszeiger (engl. Instruction-Pointer).

Die Mikroprogramme für die Ausführung der benutzten Maschinenbefehle stehen bereits im MPS (UEB3), das Mikroprogramm für die Holphase und den Anstoß der Ausführungsphase ist eine Endlosschleife und wird Interpreterzyklus genannt:



Übung 5:

- Laden Sie UEB3 in den MPS und UEB1 ins RAM.
- Entwerfen Sie das Mikroprogramm für die Holphase und den Anstoß der Ausführungsphase und speichern Sie es ab Platz 00 im MPS. Sichern Sie den neuen Inhalt des MPS unter dem Namen UEB5 in Ihrem Homeverzeichnis.
- Codieren Sie das Assemblerprogramm aus Übung 4 und schreiben Sie das Ergebnis ins RAM (ab Adresse 00). Sichern Sie den RAM-Inhalt unter dem Namen UEB5.
- Löschen Sie alle Register (Registerfenster, F9). Starten Sie die Ausführung des Maschinenprogramms durch Auslösen eines Taktes. Im Gegensatz zu einem realen Prozessor verlangt MIKROSIM nach Ende einer jeden Phase einen erneuten Anstoß.

6 Mikroprogramme für unbedingte Sprünge

Übung 6:

- a) Entwerfen Sie ein Mikroprogramm für einen Sprung an diejenige (RAM-) Adresse, die im Register DR steht: JMP nn. Achtung: JMP [nn] ist etwas anderes!
- b) Entwerfen Sie ein Mikroprogramm für einen relativen Sprung. Die Zieladresse errechnet sich dabei aus IP + nn: JR nn.
- c) Der Befehl HALT kann als Endlosschleife im Mikroprogramm realisiert werden. Wie?
- d) Laden Sie UEB5 in den MPS, schreiben Sie die in a), b) und c) entworfenen Mikroprogramme in den MPS (Start bei 42, 43 bzw. 4F), und sichern Sie den neuen Inhalt des MPS unter dem Namen UEB6 in Ihrem Homeverzeichnis.
- e) Laden Sie das Maschinenprogramm UEB5 ins RAM, fügen Sie den Code von JMP 02 oder JR -8 an und testen Sie das neue Programm. Speichern Sie es dann als UEB6.

7 Mikroprogramme für bedingte Anweisungen (Sprünge)

Bei einer bedingten Anweisung wird deren Ausführung vom Zustand eines Flags abhängig gemacht. Dies wird dadurch erreicht, dass der Zustand des Flags auf das Steuerwerk einwirkt und dort die Ermittlung der Folgeadresse im MPS beeinflusst. MIKROSIM verfügt nur über das Signflag SF, welches genau dann gesetzt ist, wenn $AX > 7F$, also als negative Zahl interpretiert werden kann.

Für MIKROSIM gilt: die Folgeadresse ergibt sich durch (interne) Addition von FA, dem durch Tor F gelieferten Wert (entweder 00 oder OP) sowie dem durch Tor E gelieferten Wert. Bisher war Tor E nie geöffnet und trug zur Berechnung der Folgeadresse deshalb immer den Summanden 0 bei. Wird das Tor E geöffnet, so gelangt der Zustand des Flags SF, also 0 oder 1, zusätzlich in das Steuerwerk.

Beispiel: Es soll das Mikroprogramm (Startadresse 50) für einen Maschinenbefehl entwickelt werden, der den Inhalt von AX genau dann um 1 vergrößert, wenn SF gesetzt ist.

Folgeadresse,
falls SF = 0:
Folgeadresse,
falls SF = 1:

Adr	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	FA
50															X		51
51																	00
52																	00

Übung 7:

- a) Laden Sie UEB6 in den MPS. Schreiben Sie Mikroprogramme für JS nn und für JNS nn mit den Startadressen 46 bzw. 49 in den MPS. Sichern Sie den neuen Inhalt des MPS als UEB7.
- b) Schieben Sie im Maschinenprogramm UEB6 vor dem Code von INC [1F] einen bedingten Sprung ein, der zum Befehl HALT führt, welcher ganz am Ende des Programms anzufügen ist. Damit ist das Maschinenprogramm zur Ganzzahldivision komplett. Speichern Sie es als UEB7 in Ihrem Homeverzeichnis und testen Sie es.

8 Interpreter für eine minimale Maschinsprache

Übung 8:

- a) Die folgende Tabelle enthält einen sinnvollen Befehlssatz für MIKROSIM, von dem die mit "✓" markierten Teile schon als Mikroprogramme in UEB7 enthalten sind. Ergänzen Sie die restlichen Befehle und sichern Sie den so entstandenen Interpreter für Maschinenprogramme unter dem Namen MININT in Ihrem Homeverzeichnis. nn steht für ein Byte (2 Nibble).

Befehl	Code	Befehl	Code	Befehl	Code	Befehl	Code
mov AX, [nn]	10 ✓	add AX, [nn]	20	inc [nn]	30 ✓	jmp [nn]	40
mov AX, nn	12	add AX, nn	22	dec [nn]	35	jmp nn	42 ✓
mov [nn], AX	13 ✓	sub AX, [nn]	23 ✓	inc AX	3A *	jr nn	43 ✓
mov BX, [nn]	16	sub AX, nn	25	dec AX	3D *	js nn	46 ✓
mov BX, nn	18	add AX, BX	28 *			jns nn	49 ✓
mov [nn], BX	19	Sub AX, BX	2B *			halt	4F *
mov AX, BX	1C *	Sub AX, AX	2E *				
mov BX, AX	1E *						

* Bei diesen Befehlen erübrigt sich die Angabe eines Operanden, das zweite Byte des Befehlscodes hat also keine Bedeutung. Wegen der für alle Befehle gleichen Hohlphase muss es aber dennoch vorhanden sein.

(* Alternativ könnten diese Befehle als echte 1-Byte-Befehle implementiert werden. Da die Hohlphase für 2-Byte-Befehle angelegt ist, müsste hier vor der eigentlichen Befehlsausführung erst IP um 1 reduziert werden!)

- b) Das nebenstehende Assemblerprogramm soll RAM[1E] mit RAM[1F] multiplizieren und das Ergebnis nach RAM[1C] schreiben. RAM[1D] dient als Hilfsspeicher. Assemblieren Sie dieses Programm "von Hand", so dass es vom Interpreter MININT aus a) abgearbeitet werden kann. Bringen Sie das assemblierte Programm ins RAM, sichern Sie es unter dem Namen UEB8b und testen Sie es.

```

mov AX, [1E]
mov [1D], AX
sub AX, AX
mov [1C], AX
M1: mov AX, [1D]
    dec AX
    js M2
    mov [1D], AX
    mov AX, [1C]
    add AX, [1F]
    mov [1C], AX
    jmp M1
M2: halt

```

- c) MIKROSIM soll mit Hilfe des Interpreters MININT aus a) den ggT der Zahlen a und b nach dem folgendem Algorithmus berechnen. Schreiben Sie das Assemblerprogramm, assemblieren Sie es, sichern Sie es als UEB8c und testen Sie es ausführlich. (a = RAM[1E], b = RAM[1F], ggT = RAM[1F])

```

while a <> b do
    if a < b then b := b - a
    else a := a - b

```

Bemerkung für Leute, die unbedingt einen Assembler schreiben möchten:

Für die Dateien *.RAM gilt:

```

Dateityp = FILE OF RAMTyp;
RAMTyp = ARRAY [$00..$FF] OF byte;

```

9 Verschiedene Adressierungsarten

Die wichtigsten Adressierungsarten sind in der folgenden Übersicht zusammen-gestellt:

Adressierungsart	Operand im	Adressangabe	Beispiel
unmittelbar	Befehl	unnötig	mov AX, 07
Register	Register	im Befehl	mov AX, BX
Speicher			
direkt	Speicher	im Befehl	mov AX, [07]
indirekt	Speicher	im Register	mov AX, [BX]
relativ	Speicher	im Speicher	Mov AX, [BX+07]

Anregungen:

Entwerfen Sie für die Transport-, Rechen- und Sprungbefehle auch Varianten für die indirekte und die relative Adressierung. Bezugsregister soll dabei BX sein. Damit lassen sich mit MIKROSIM dann auch Felder bearbeiten!

10 Maschinenbefehle für Stackoperationen

Das Register BX wurde für den Interpreter MININT aus 8a) nicht unbedingt benötigt und kann deshalb als Stackpointer verwendet werden. Der Stack soll am oberen Ende des RAM liegen und zu kleinen Adressen hin wachsen. Hierzu ist BX mit dem Wert 00 zu initialisieren (Assemblerbefehl ins).

Anregungen für weitere Stackbefehle:

push AX	ads { addiert die obersten Stackelemente }	call b
pop AX	sub { subtr. " " " }	ret

{Schönheitsfehler bei call: AX wird als Zwischenspeicher gebraucht!}

weitere Beispiele in:

Arbeitshefte Informatik

Vom Programm zum Prozessor

von Werner Simon

Ernst Klett Schulbuchverlag Stuttgart, 1994

ISBN 3-12-717760-7

B Lösungen

Lösung zu Übung 8a:

AD	Befehl	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	FA
00	mov OP, [IP]						5			8								01
01														C				02
02								6			9							03
03	inc IP					4	5		7									04
04	mov DR, [IP]						5			8								05
05														C				06
06	inc IP					4	5		7									07
07	do command																F	00
08																		09
09																		0A
0A																		0B
0B																		0C
0C																		0D
0D																		0E
0E																		0F
0F																		10
10	mov AX, [nn]									8	9							11
11														C				12
12	mov AX, nn	0									9							00
13	mov [nn], AX									8	9							14
14			1									A						15
15													B					00
16	mov BX, [nn]									8	9							17
17														C				18
18	mov BX, nn			2							9							00
19	mov [nn], BX									8	9							1A
1A					3							A						1B
1B													B					00
1C	mov AX, BX	0			3													00
1D																		1E
1E	mov BX, AX		1	2														00
1F																		20
20	add AX, [nn]									8	9							21
21														C				22
22	add AX, nn	0	1								9							00
23	sub AX, [nn]									8	9							24
24														C				25
25		0	1								9				D			00
26																		27
27																		28
28	add AX, BX				3							A						29
29		0	1								9							00
2A																		2B
2B	sub AX, BX				3							A						2C
2C		0	1								9				D			00
2D																		2E
2E	sub AX, AX		1									A						2F
2F		0	1								9				D			00

AD	Befehl	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	FA
30	inc [nn]									8	9							31
31														C				32
32				2							9							33
33					3				7			A						34
34													B					00
35	dec [nn]									8	9							36
36														C				37
37				2							9							38
38					3				7			A			D			39
39														C				00
3A	inc AX	0	1						7									00
3B																		3C
3C																		3D
3D	dec AX	0	1						7						D			00
3E																		3F
3F																		40
40	jmp [nn]									8	9							41
41														C				42
42	jmp nn					4					9							00
43	jr nn					4	5				9							00
44																		45
45																		46
46	js nn															E		47
47																		00
48						4					9							00
49	jns nn															E		4A
4A						4					9							00
4B																		00
4C																		4D
4D																		4E
4E	ende	0		2		4		6		8		A						4F
4F	halt					4		6										4F

Lösung zu Übung 8b:

RAM[1C] := RAM[1E] * RAM[1F],
RAM[1D] ist Hilfsspeicher

Assemblerprogramm	Maschinenprogramm
mov AX,[1E]	00 10 1E
mov [1D],AX	02 13 1D
sub AX,AX	04 2E 00
mov [1C],AX	06 13 1C
M1: mov AX,[1D]	08 10 1D
dec AX	0A 3D 00
js M2	0C 46 18
mov [1D],AX	0E 13 1D
mov AX,[1C]	10 10 1C
add AX,[1F]	12 20 1F
mov [1C],AX	14 13 1C
jmp M1	16 42 08
M2: halt	18 4F 00

Lösung zu Übung 8c:

RAM[1F] := ggT(RAM[1E],RAM[1F])

Assemblerprogramm	Maschinenprogramm
M1: mov AX,[1E]	00 10 1E
sub AX,[1F]	02 23 1F
js KL	04 46 0C
dec AX	06 3D 00
jns GR	08 49 14
halt	0A 4F 00
KL: mov AX,[1F]	0C 10 1F
sub AX,[1E]	0E 23 1E
mov [1F],AX	10 13 1F
jmp M1	12 42 00
GR: inc AX	14 3A 00
mov [1E],AX	16 13 1E
jmp M1	18 42 00

C Weitere Aufgaben

Fragen zu MIKROSIM (Klassenarbeit!)

1. Welcher Unterschied besteht zwischen den Inhalten von OP (OperationPointer) und IP (InstructionPointer)?
2. Der IP wird häufig als Programmzähler bezeichnet. Was ist davon zu halten?
3. Im RAM befindet sich der Code des folgenden Assemblerprogramms für MININT

```
00  mov AX, [06]
02  sub AX, 20
04  mov [06], AX
06  jmp 00
08  halt
```

- a) Welche Belegung hat das RAM, nachdem 4 Befehle ausgeführt wurden?
 - b) Hält das Programm an?
4. Wovon hängt es ab, ob der Inhalt einer RAM-Zelle bei der Ausführung eines Maschinenprogramms als Befehlscode, als Adresse oder als Datum interpretiert wird?
 5. Warum lässt sich der CMP-Befehl nicht mit MIKROSIM realisieren?
Wie könnte der Simulator zu diesem Zweck ergänzt werden?