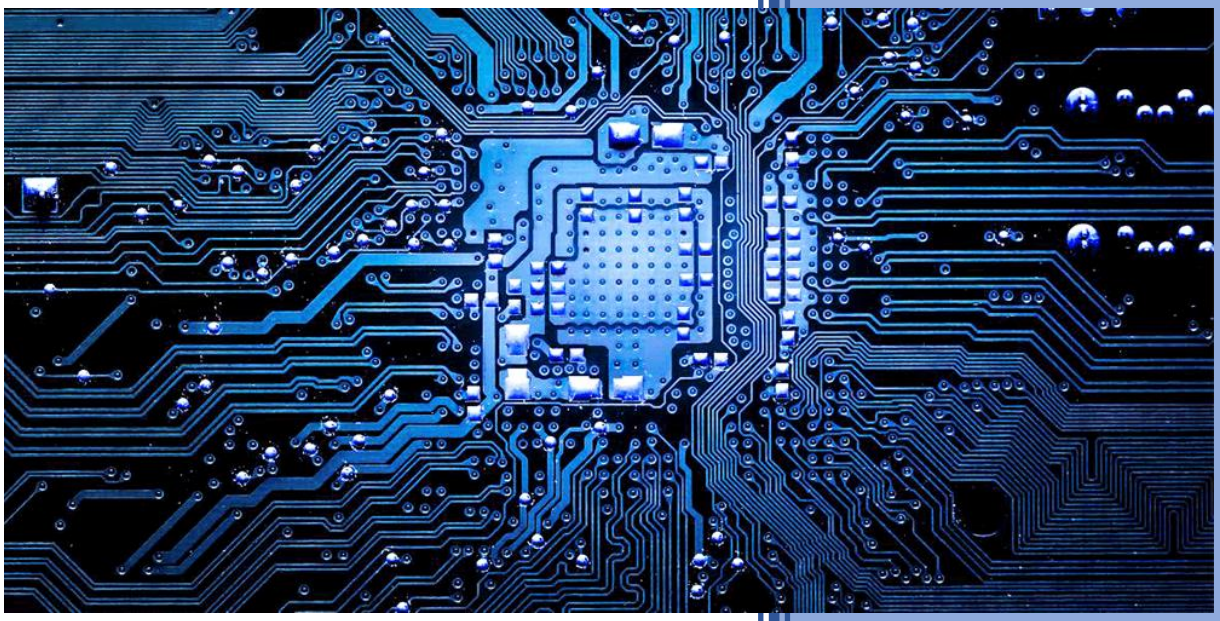
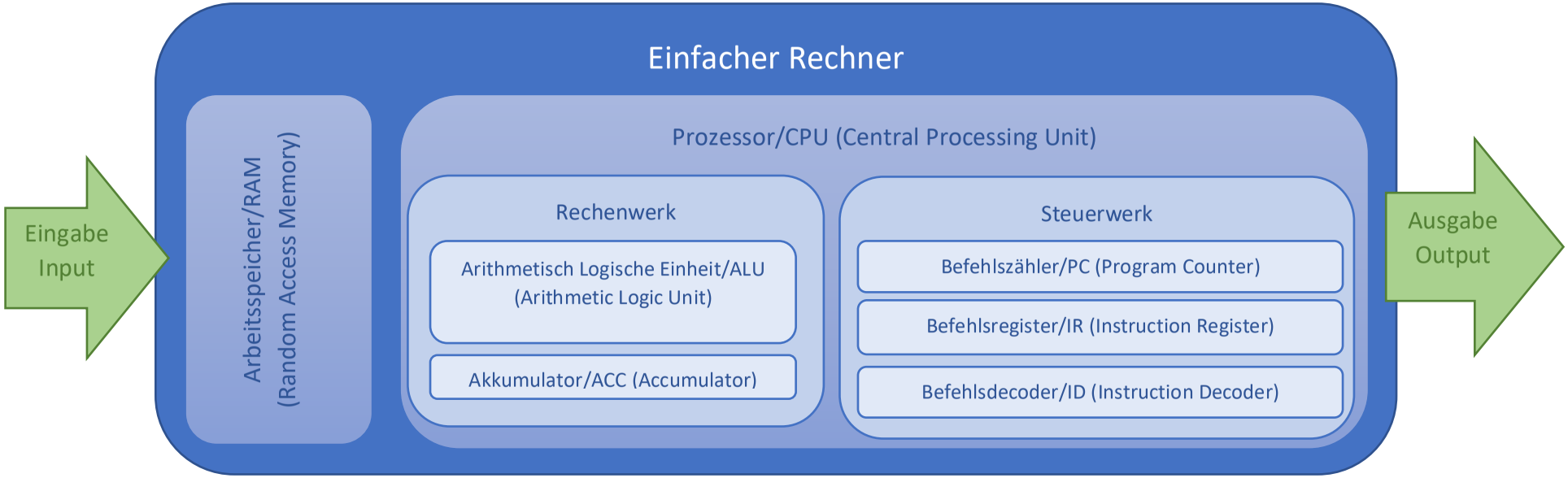
Funktionsweise eines Rechners



Carmen Christen

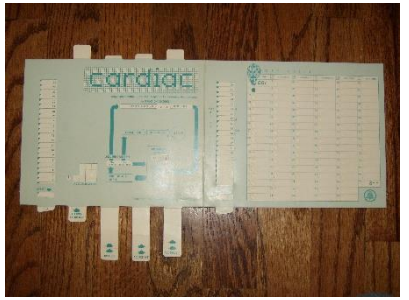
05.10.2020

Die Komponenten eines Rechners und deren Funktionen



Komponente	*Funktion*
RAM	Der RAM speichert die von der CPU gerade auszuführenden Programme oder Programmteile und die dabei benötigten Daten.
Rechenwerk	Im Rechenwerk werden Daten entsprechend den Operationsanweisungen des Steuerwerks verknüpft oder bearbeitet.
ALU	In der ALU werden die Operationen ausgeführt. Sie bildet das Kernstück eines Rechenwerks. Nebst der ALU enthält das Rechenwerk Hilfs- und Steuerregister, wie den Akkumulator.
Akkumulator	Der Akkumulator ist meist direkt mit der ALU verbunden und speichert deren Ergebnisse.
Steuerwerk	Das Steuerwerk steuert den Ablauf der Befehlsverarbeitung. Dazu dekodiert es den im Befehlsregister stehenden Befehl und generiert daraus die erforderlichen Operationsanweisungen für das Rechenwerk.
PC	Der Befehlszähler speichert die Adresse des als nächstes auszuführenden Befehls.
IR	Das Befehlsregister ist ein Zwischenspeicher für den aktuell auszuführenden Maschinenbefehl.
ID	Der ID übersetzt die Maschinenbefehle und zerlegt diese, falls nötig, in einzelne Arbeitsschritte, welche dann an die zugehörigen Einheiten der CPU weitergeleitet werden. („Die Befehle werden in einer Tabelle nachgeschlagen.“)

CARDIAC (CARDboard Illustrative Aid to Computation)



Als CARDIAC entwickelt wurde, war der Zugang zu Computern sehr beschränkt. Daher entwickelte David Hagelbarger CARCIAC als eine illustrative Hilfe aus Karton um die Funktionsweise eines Rechners zu unterrichten. Später wurde dann zusätzlich ein CARDIAC-Simulator entwickelt. Diesen werden wir im Folgenden verwenden, um so unsere Programme jeweils auch gleich testen zu können.

CARDIAC vs. Realer Rechner: In einem richtigen Rechner sind sowohl die Speicheradressen als auch deren Inhalte im Binärcode dargestellt. Da hohe Binärzahlen allerdings relativ viele Stellen benötigen, werden hier der Einfachheit halber die Speicheradressen und deren Inhalte als Dezimalzahlen dargestellt.

Memory

00: 001	10:	20:	30:	40:	50:	60:	70:	80:	90:
01:	11:	21:	31:	41:	51:	61:	71:	81:	91:
02:	12:	22:	32:	42:	52:	62:	72:	82:	92:
03:	13:	23:	33:	43:	53:	63:	73:	83:	93:
04:	14:	24:	34:	44:	54:	64:	74:	84:	94:
05:	15:	25:	35:	45:	55:	65:	75:	85:	95:
06:	16:	26:	36:	46:	56:	66:	76:	86:	96:
07:	17:	27:	37:	47:	57:	67:	77:	87:	97:
08:	18:	28:	38:	48:	58:	68:	78:	88:	98:
09:	19:	29:	39:	49:	59:	69:	79:	89:	99: 8--

Deck | **Reader** | **CPU** | **Output**

PC: 00

Load

Instruction Decoder

Instruction Register:

Opcode: Operand:

Accumulator: 0000

Reset Clear Mem

Step Slow Run Halt

Befehle in CARDIAC

OC = Operationscode
ACC = Akkumulator
PC = Program Counter (Befehlszähler)

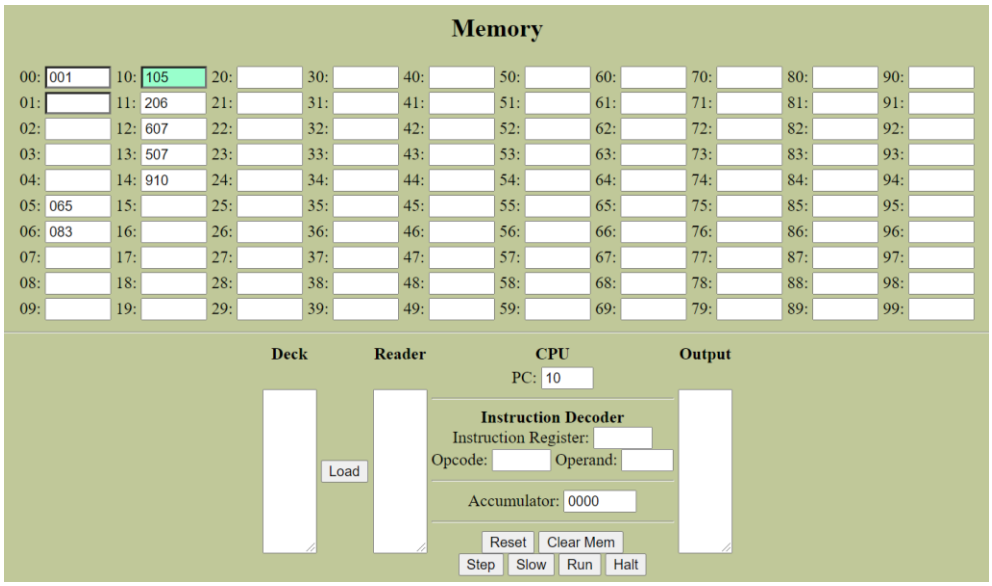
OC	*Abkürzung*	*Operation*
0	INP (Input)	Schreibe den nächsten Befehl vom Deck in die angegebene Speicheradresse.
1	CLA (Clear ACC)	Setze den ACC auf null und addiere die Zahl der angegebenen Speicheradresse.
2	ADD (Add to ACC)	Addiere die Zahl der angegebenen Speicheradresse zur Zahl im ACC.
3	TAC (Test ACC)	Teste, ob die Zahl im ACC negativ ist. Falls ja springe zur angegebenen Adresse.
4xy	SFT (Shift ACC)	Schiebe zuerst den Inhalt von ACC um x Stellen nach links (und fülle rechts mit Nullen auf) und danach um y Stellen nach rechts (und fülle links mit Nullen auf).
5	OUT (Output)	Gib den Inhalt der angegebenen Adresse aus.
6	STO (Store ACC)	Speichere die Zahl im ACC unter der angegebenen Adresse.
7	SUB (Subtract form ACC)	Subtrahiere die Zahl der angegebenen Adresse von der Zahl im ACC.
8	JMP (Jump)	Erhöhe den PC um eins und speichere 800 + PC unter der Adresse 99. Kopiere die angegebene Adresse in den PC. (Springe zur angegebenen Adresse.)
9	HRS (Halt and reset)	Stoppe das Programm und setzte den PC auf die angegebene Adresse.

Symbole in Flussdiagrammen

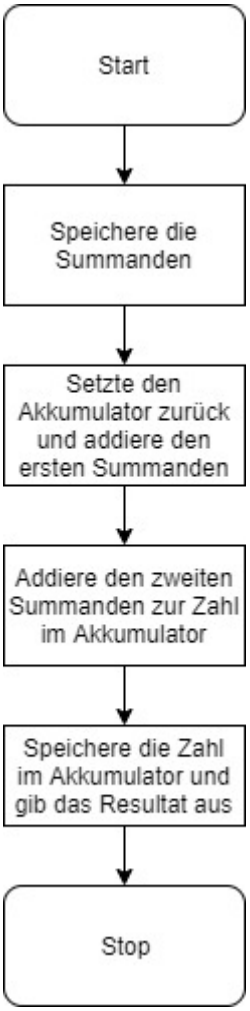
Die unterschiedlichen Symbole werden in der gewünschten Reihenfolge mit Pfeilen verbunden. Dabei zeigt der Pfeil jeweils auf das nächste auszuführende Symbol. Bei Entscheidungen können zwei unterschiedliche Pfeile, ein „Ja“ und ein „Nein“ Pfeil, die Raute verlassen. Pfeile können auch zu einem bereits ausgeführten Symbol zurückzeigen, um so eine Schleife zu erzeugen.

Beispiel 1: Addition von zwei Zahlen (65+83)

Vor dem Ausführen deines Programmes sollte der Simulator ungefähr folgendes anzeigen.

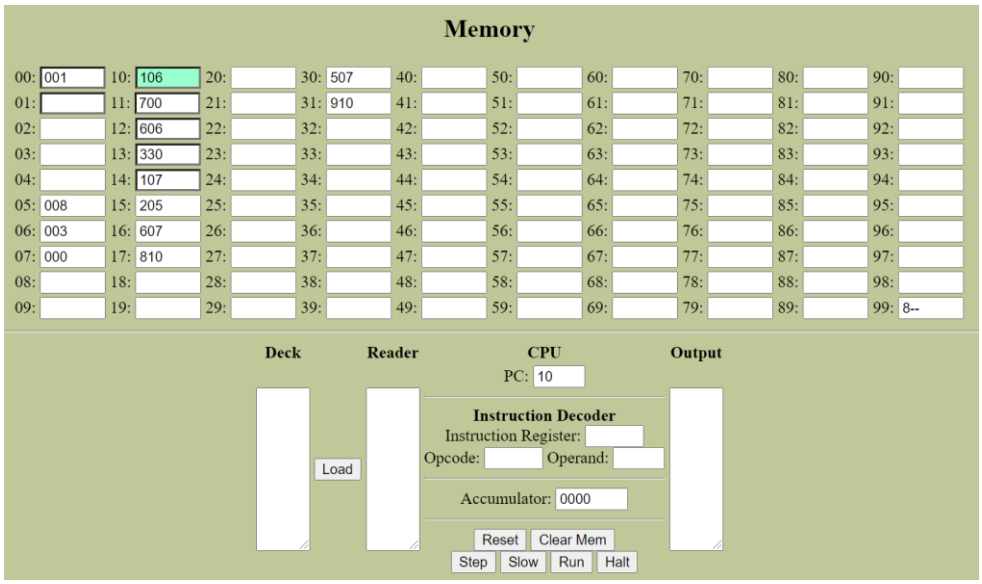


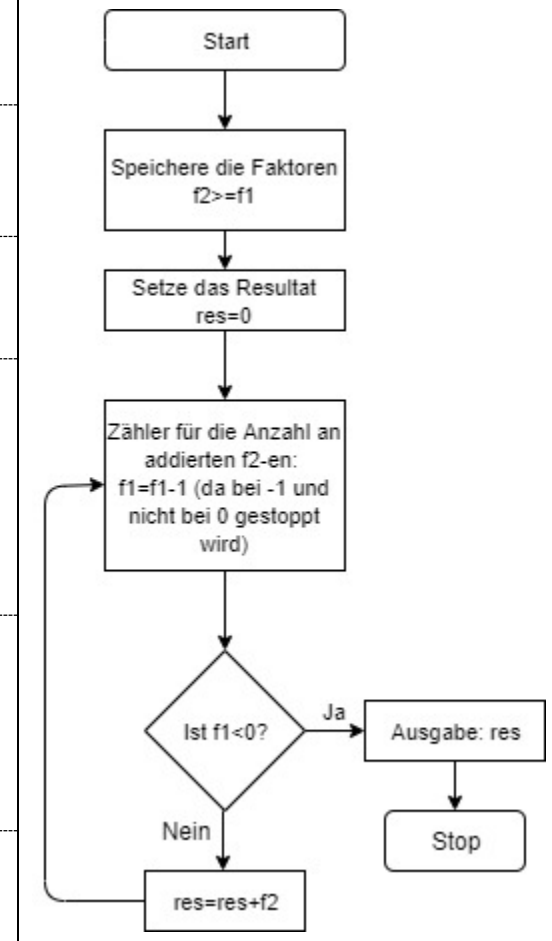
Adresse	*Inhalt*	*Kommentare*	*Flussdiagramm*
		Setzte den PC auf 10.	
05 06	065 083	Der erste Summand (65) wird unter der Adresse 05 gespeichert und der zweite Summand (83) wird unter der Adresse 06 gespeichert.	
10	105	Die Programmanweisungen beginnen bei der Adresse 10. Die erste Anweisung setzt den ACC auf null und addiert den Inhalt von Adresse 05.	
11	206	Der Inhalt von Adresse 06 wird zur Zahl im ACC dazu addiert.	
12 13	607 507	Speichere die Zahl im ACC unter der Adresse 07. Gib den Inhalt von Adresse 07 aus.	
14	910	Das Programm wird beendet und der PC wird wieder auf 10 zurückgesetzt.	



Beispiel 2: Multiplikation von zwei Zahlen (8 · 3)

Vor dem Ausführen deines Programmes sollte der Simulator ungefähr folgendes anzeigen.



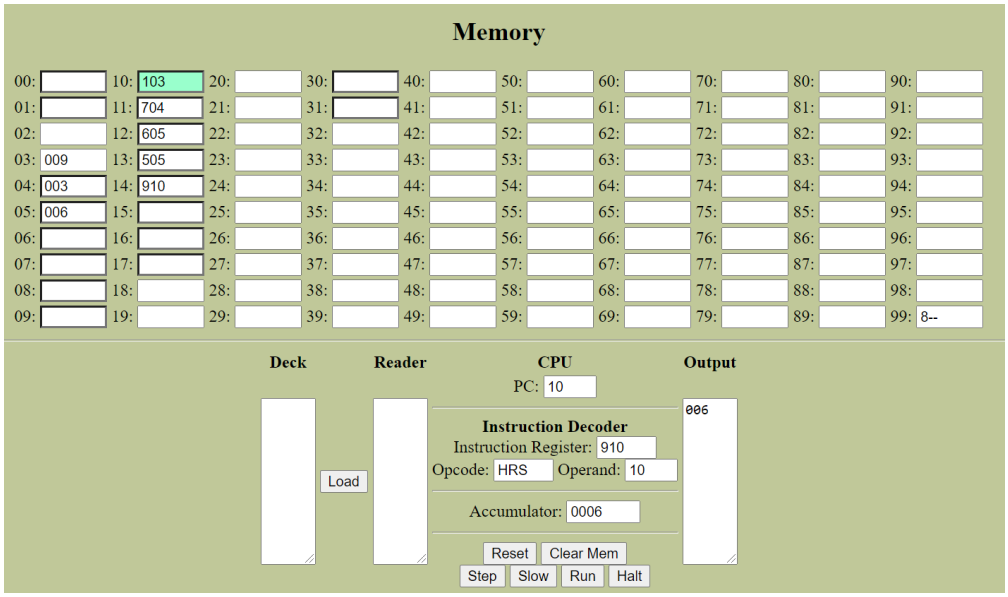
Adresse	*Inhalt*	*Kommentare*	*Flussdiagramm*
00	001	Speichert die 001 in Zelle 00. Setze den PC auf 10.	
05 06	008 003	Speichert den grösseren Faktor (8) unter der Adresse 05 und den kleineren Faktor (3) unter der Adresse 06.	
07	000	Das Resultat wird unter der Adresse 07 gespeichert und zu Beginn auf null gesetzt.	
10 11 12	106 700 606	Die Programmanweisungen beginnen bei Adresse 10. Setzt den ACC auf null, addiert den Inhalt von Adresse 06 (kleinerer Faktor), subtrahiert den Inhalt der Adresse 00, also eins und speichert das Resultat wieder unter der Adresse 06.	
13 30 31	330 507 910	Testet, ob die Zahl im ACC negativ ist. Falls ja wird der PC auf 30 gesetzt, das Resultat ausgegeben, das Programm beendet und der PC wieder auf 10 zurückgesetzt. Falls nicht, wird der PC um eins erhöht.	
14 15 16 17	107 205 607 810	Lädt den Inhalt von Adresse 07 in den ACC, addiert den Inhalt von Adresse 05 und speichert das Resultat wieder unter der Adresse 07. Anschliessend wird der PC auf 10 gesetzt.[1]	

[1] Für einen allfälligen Rücksprung (wird hier nicht benötigt) auf die folgende Zelle 18 wird ein entsprechender JMP-Befehl unter der Adresse 99 gespeichert.

Aufgabe 1: Subtraktion

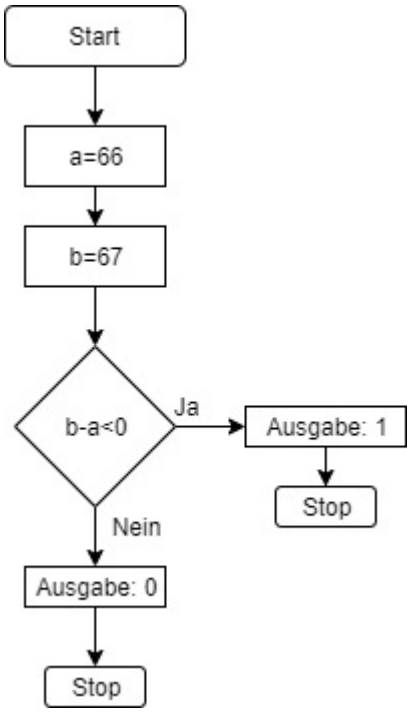
Erstelle ein Programm, welches die Zahl in Zelle 04 von der Zahl in Zelle 03 subtrahiert.

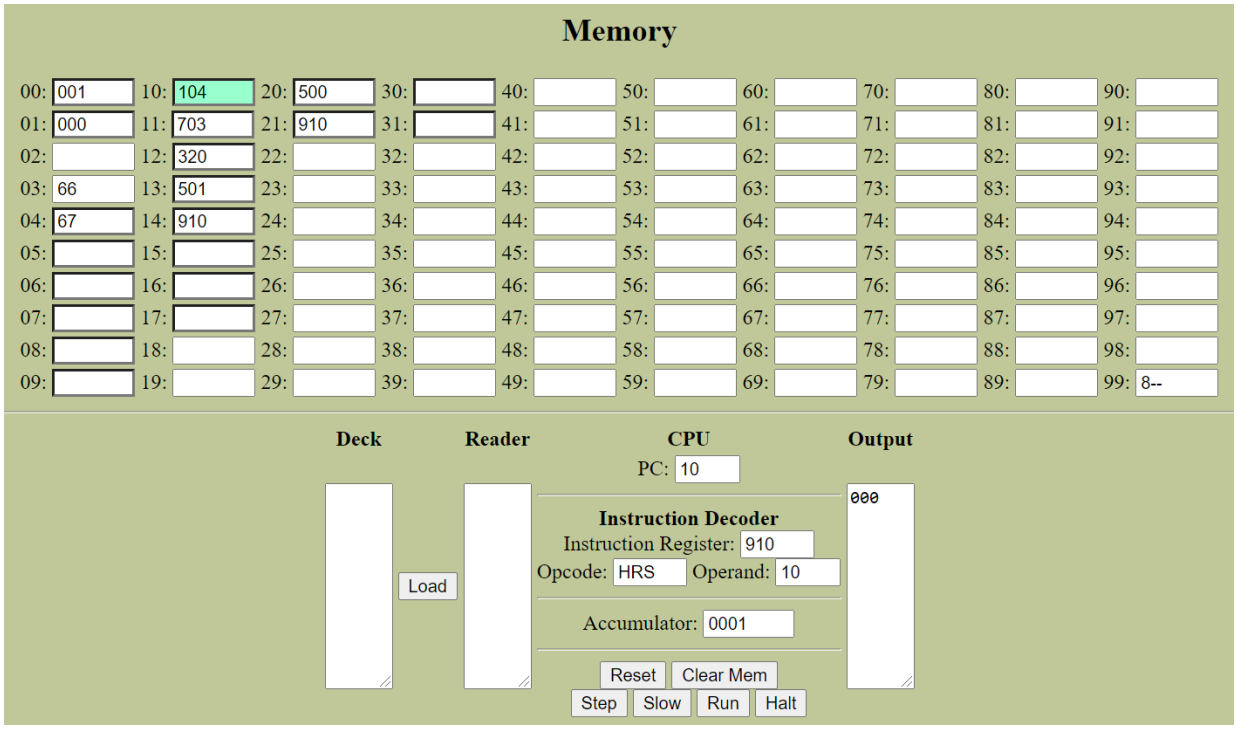
Adresse	*Inhalt*	*Kommentare*
03 04	009 006	Speichert den Minuenden (009) in Zelle 03 und den Subtrahenden (006) in Zelle 04.
10 11 12 13 14	103 704 605 505 910	Leert den ACC und lädt den Inhalt von Zelle 03 in den ACC. Subtrahiert den Inhalt von Zelle 04 von der Zahl im ACC. Speichert die Zahl im ACC unter der Adresse 05. Gibt den Inhalt von Adresse 05 aus. Beendet das Programm und setzt den PC auf 10 zurück.



Aufgabe 2: Bedingung

Erstelle ein Programm, welches 1 ausgibt, falls die Zahl a in Zelle 03 grösser ist als die Zahl b in Zelle 04, und sonst 0. Erstelle dazu zuerst ein Flussdiagramm.

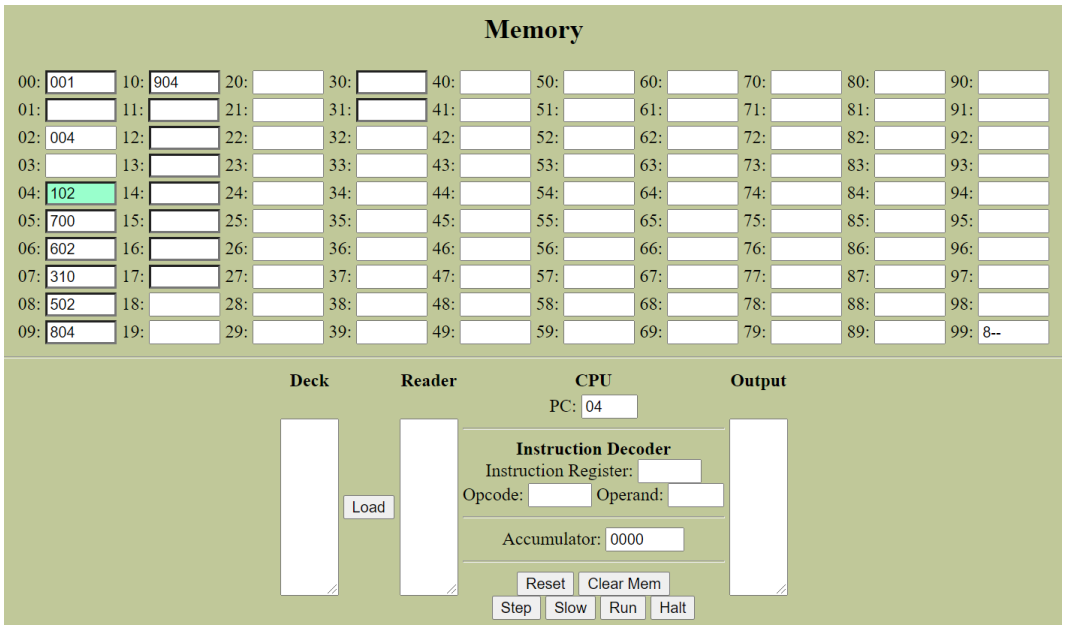
Adresse	*Inhalt*	*Kommentare*	*Flussdiagramm*
00 01	001 000	Setze den PC auf 10. Speichert die 1 unter der Adresse 00 und 0 unter der Adresse 01.	
03	066	Speichert die Zahl a (66) unter der Adresse 03.	
04	067	Speichert die Zahl b (67) unter der Adresse 04.	
10 11 12 13 14 20 21	104 703 320 501 910 500 910	Lädt den Inhalt von Zelle 04 in den ACC. Subtrahiert den Inhalt von Zelle 03. Testet, ob die Zahl im ACC negativ ist. Falls nein wird der Inhalt von Zell 01, also 0 ausgegeben und falls ja, wird der Inhalt von Zelle 00, also 1 ausgegeben. In beiden Fällen wird das Programm beendet und der PC auf 10 zurückgesetzt.	

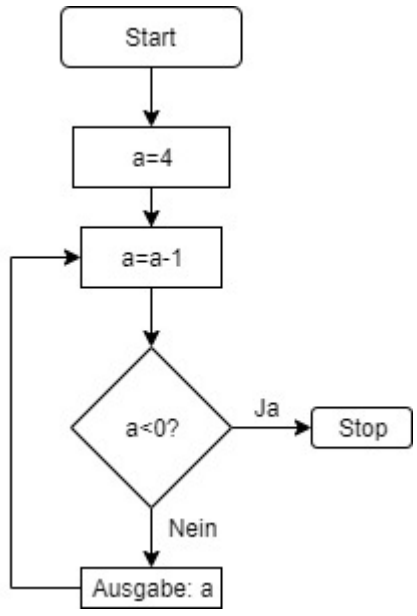


Aufgabe 3: Was macht das folgende Programm?

Ergänze die Kommentare und zeichne anschliessend das passende Flussdiagramm. Was wird ausgegeben?

Das Programm gibt die Zahlen 3, 2, 1, 0 aus.

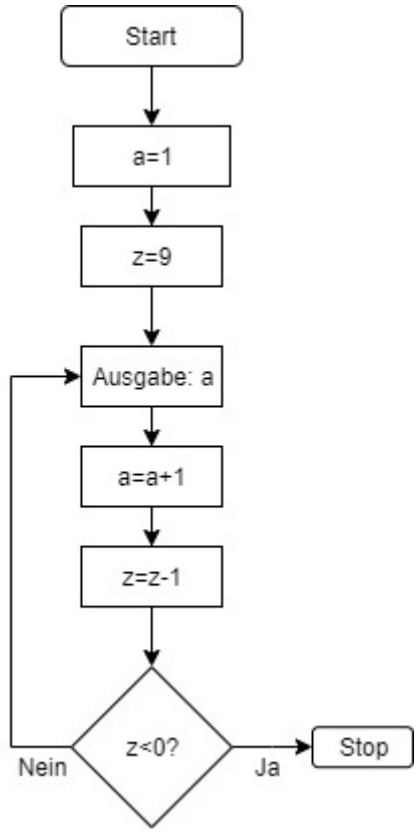


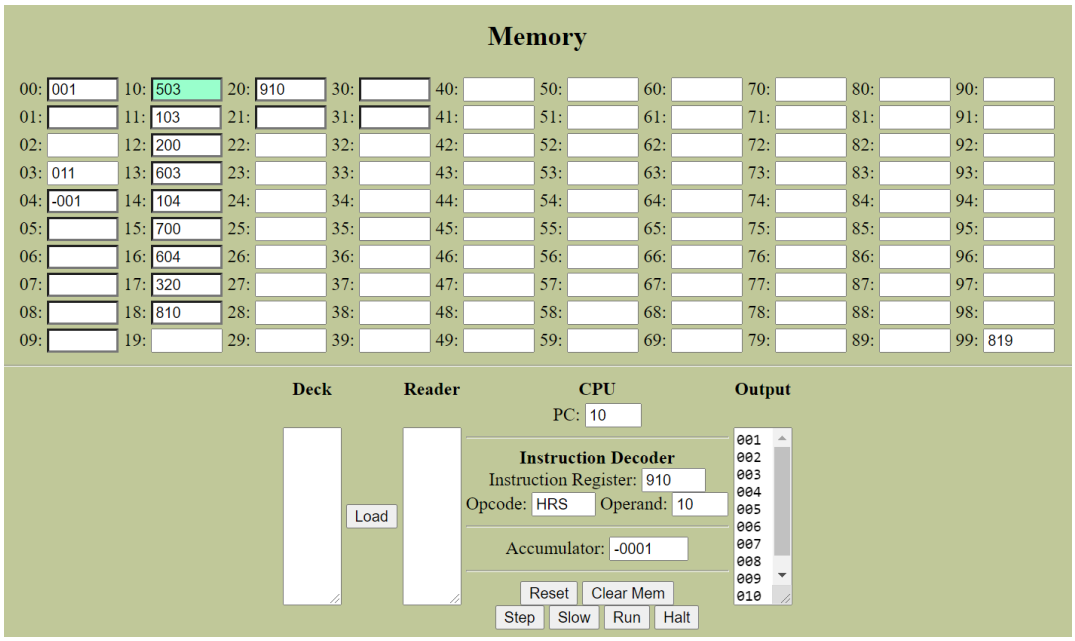
Adresse	*Inhalt*	*Kommentare*	*Flussdiagramm*
00	001	Setze den PC auf 04. Speichert die Zahl 1 unter der Adresse 00.	
02	004	Speichert die Zahl 4 unter der Adresse 02.	
04 05 06	102 700 602	Lädt den Inhalt von Zelle 02 in den ACC, subtrahiert den Inhalt von Zelle 00, also 1 und speichert das Resultat in Zelle 02.	
07 08 09 10	310 502 804 904	Testet, ob die Zahl im ACC negativ ist. Falls nein, so wird der Inhalt von Zelle 02 ausgegeben und der PC auf 04 gesetzt[2]. Falls ja, so wird das Programm beendet und der PC auf 04 gesetzt.	

[2] Für einen allfälligen Rücksprung (wird hier nicht benötigt) auf die folgende Zelle 10 wird ein entsprechender JMP-Befehl unter der Adresse 99 gespeichert.

Aufgabe 4: 1-10

Erstelle ein Programm, welches die Zahlen von 1 bis 10 ausgibt. Erstelle dazu zuerst ein Flussdiagramm.

Adresse	*Inhalt*	*Kommentare*	*Flussdiagramm*
00	001	Setzte den PC auf 10. Speichert die Zahl 1 in Zelle 00.	
03	001	Speichert die Zahl 1 in Zelle 03.	
04	009	Speichert den Zähler z=9 in Zelle 04.	
10	503	Gibt den Inhalt von Zelle 03 aus.	
11 12 13	103 200 603	Erhöht den Inhalt von Zelle 03 um eins.	
14 15 16	104 700 604	Senkt den Inhalt von Zelle 04 um eins.	
17 18 20	320 810 910	Testet, ob der Inhalt (Zähler z) im ACC negativ ist. Falls nicht, wird der PC auf 10 gesetzt.[3] Falls ja, wird das Programm beendet und der PC auf 10 zurückgesetzt.	

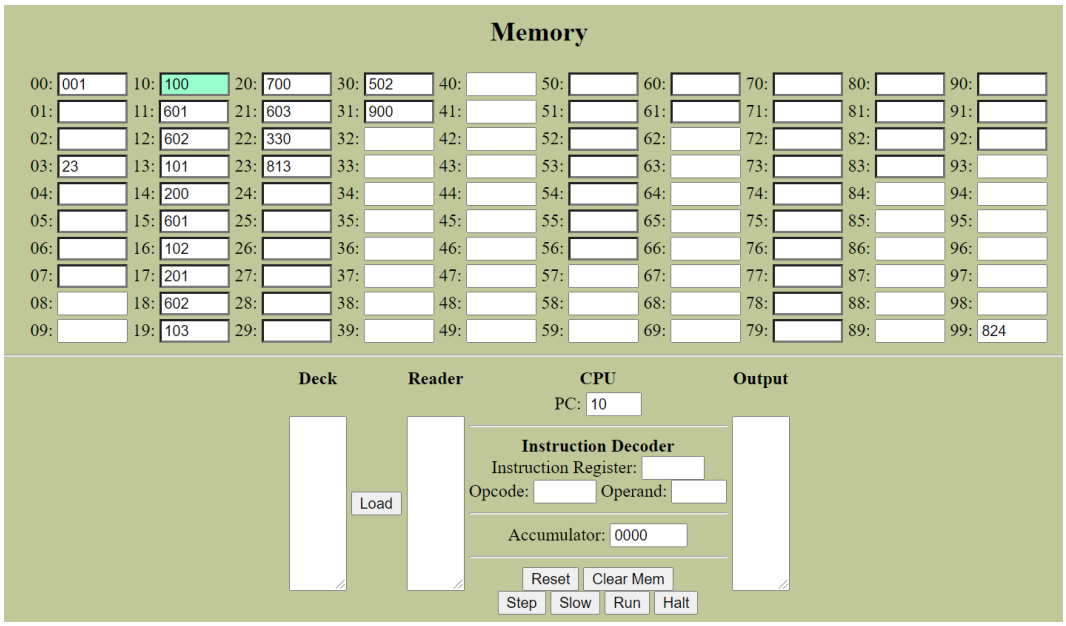


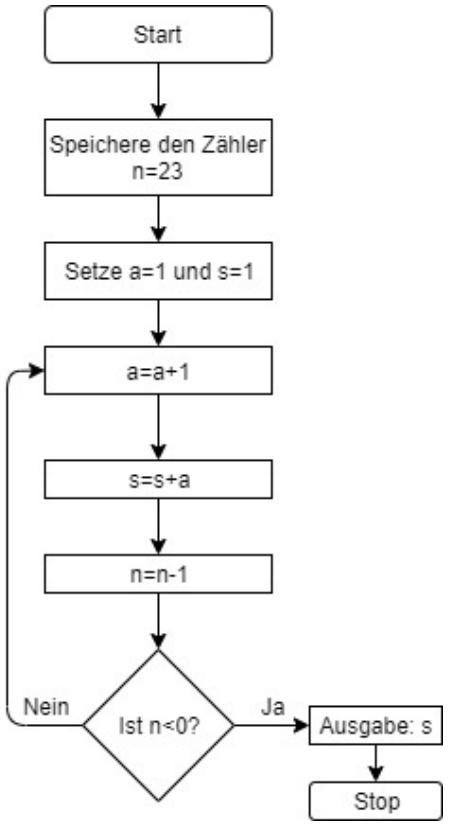
[3] Für einen allfälligen Rücksprung (wird hier nicht benötigt) auf die folgende Zelle 19 wird ein entsprechender JMP-Befehl unter der Adresse 99 gespeichert.

Aufgabe 5: Was macht das folgende Programm?

Ergänze die Kommentare und zeichne anschliessend das passende Flussdiagramm. Was wird ausgegeben?

Das untenstehende Programm berechnet die Summe 1+2+…+25 und gibt das Resultat 325 aus.



Adresse	*Inhalt*	*Kommentare*	*Flussdiagramm*
00	001	Speichert die 001 unter der Adresse 00. Setzte den Befehlszähler auf 10.	
03	023	Speichert 23 (Zähler) unter der Adresse 03.	
10 11 12	100 601 602	Kopiert den Inhalt von Adresse 00 in die Adressen 601 und 602.	
13 14 15	101 200 601	Erhöht den Inhalt von Adresse 01 um 1.	
16 17 18	102 201 602	Addiert zum Inhalt von Adresse 02 den Inhalt von Adresse 01 und speichert das Resultat unter der Adresse 02.	
19 20 21	103 700 603	Subtrahiert 1 vom Inhalt von Adresse 03 und speichert das Resultat wieder in Adresse 03.	
22 30 31 23	330 502 900 813	Testet, ob die Zahl im Akkumulator negativ ist. Falls ja wird der Inhalt von Adresse 02 ausgegeben, das Programm beendet und der Befehlszähler auf 00 gesetzt. Falls nicht wird der Befehlszähler auf 13 gesetzt.	

Fazit

Aufgabe: Ergänze den nachfolgenden Lückentext.

Wieso haben wir uns mit der Binärdarstellung von Zahlen und den arithmetischen Grundoperationen auf Binärzahlen auseinandergesetzt? Was hat das mit Informatik zu tun?

In den letzten Wochen haben wir uns mit der **Binärdarstellung** von Zahlen beschäftig. Dies liegt daran, dass ein Computer nur zwischen zwei Zuständen unterscheiden kann. Um das möglichst einfach darzustellen, repräsentieren wir den einen Zustand mit **0** und den anderen Zustand mit **1**. Damit ein Computer Informationen (nicht nur Zahlen) oder Programme „verstehen" kann, müssen diese also zuerst in Binärcode übersetzt werden. Dies wird von einem sogenannten Compiler oder Interpreter gemacht. Um Informationen zu verarbeiten, stehen einem Rechner nur wenige Operationen zur Verfügung, beispielsweise **arithmetische** Operationen, wie die Addition und Subtraktion von Binärzahlen oder **logische** Operationen, wie die Negation. Im Grunde kann also die Funktionsweise eines Rechners auf einige wenige **arithmetische** und **logische** Operationen mit Binärzahlen zurückgeführt werden. Welche Operationen einem Rechner konkret zur Verfügung stehen, hängt von der Hardware des Rechners ab.

Wie funktioniert ein Prozessor?

Wird ein Programm ausgeführt, wird es als erstes von einem Compiler oder einem Interpreter in Binärcode, oder besser gesagt in eine Reihe von Maschinenbefehlen, übersetzt und anschliessend in den Arbeitsspeicher geladen. Das **Steuerwerk** steuert nun den Ablauf der Befehlsverarbeitung. Der **Befehlszähler** zeigt auf die Adresse der als nächstes auszuführenden Operation, welche nach dem Übersetzen nun als Maschinenbefehl, also in der Maschinensprache, vorliegt. Der als nächstes auszuführende Maschinenbefehl wird ins **Befehlsregister** geladen und muss anschliessend vom **Befehlsdecoder** in die entsprechende Operation und den entsprechenden Operanden übersetzt werden. Anschliessend werden die Arbeitsschritte zum Ausführen der Operation vom Steuerwerk an die entsprechenden Einheiten weitergeleitet.

Das Verarbeiten oder Verknüpfen von Daten gemäss den Operationsanweisungen des Steuerwerks findet im **Rechenwerk** statt. Die Resultate werden jeweils im **Akkumulator** gespeichert.

Maschinensprache-Assemblersprache-Programmiersprache

Maschinensprache wird meistens als Binärcode, mit Nullen und Einsen oder vereinfacht mithilfe von Hexadezimalzahlen dargestellt. Um das Programmieren einfacher zu machen, können Programme in einer **Assemblersprache** geschrieben werden. In **Assemblersprachen** muss man die Befehle nicht mehr im Binärcode angeben, sondern man kann einfache Befehle wie ADD oder SUB verwenden. Die Operanden können je nach **Assemblersprache** als Text, Dezimalzahl, Hexadezimalzahl oder Binärzahl angegeben werden. Um zu unterscheiden, ob es sich bei einem Operanden um eine Adresse oder eine Zahl handelt wird ein # vor Zahlen gesetzt. Damit ein **Assemblerprogramm** ausgeführt werden kann, muss es zuvor von einem **Assembler** in Maschinensprache übersetzt werden. Auch in **Assemblersprachen** ist das Programmieren von komplexeren Programmen immer noch sehr aufwendig. Programme werden oft sehr gross, was einen

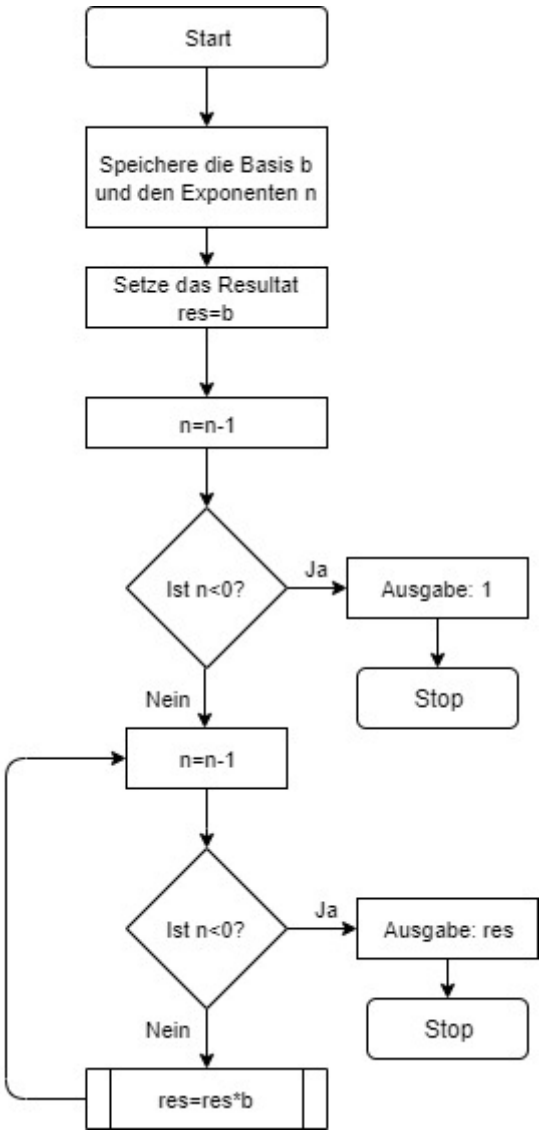
Befehl	Num	Operand	Assembler	Bedeutung
001	1	000010	LOAD #2	Lade den Wert 2 in ACC.
010	0	001101	STORE 13	Speichere ACC in Speicherzelle 13.
001	1	000101	LOAD #5	Lade den Wert 5 in ACC.
010	0	001110	STORE 14	Speichere ACC in Speicherzelle 14.
001	0	001101	LOAD 13	Lade Speicherzelle 13 in ACC.
011	0	001110	ADD 14	Addiere Speicherzelle 14 zu ACC hinzu.
010	0	001111	STORE 15	Speichere ACC in Speicherzelle 15.
111	0	000000	HALT	Programm beenden.

Überblick erschwert. Um das Implementieren solcher Programme zu erleichtern, gibt es höhere **Programmiersprachen** wie Python oder Java. Diese müssen vor dem Ausführen, wie bereits erwähnt, von einem Compiler oder einem Interpreter in Maschinensprache übersetzt werden.

```
void main(void) {
    int x = 2;
    int y = 5;
    int z = x + y;
}
```

Zusatzaufgabe: Potenzieren

Erstelle ein Programm, welches eine Potenz b^n berechnet. Das Flussdiagramm ist bereits gegeben. Ergänze die fehlenden Einträge in der Tabelle.

Adresse	*Inhalt*	*Kommentare*	*Flussdiagramm*
00 01	001 000	Speichert die 001 unter Adresse 00 und die 000 unter Adresse 01. Setzt den Befehlszähler auf 10!	
05 06	002 004	Speichert die Basis b=2 unter Adresse 05 und den Exponenten n=4 unter Adresse 06.	
07	002	Speichert das Resultat res=b=2 unter Adresse 07.	
10 11 12	106 700 606	Lädt den Inhalt von Adresse 06 in den Akkumulator, subtrahiert den Inhalt von Adresse 00 und speichert das Resultat unter der Adresse 06.	
13 40 41	340 500 900	Testet, ob die Zahl im Akkumulator negativ ist. Falls ja wird der Befehlszähler auf 40 gesetzt, der Inhalt von Adresse 00 ausgegeben, sowie das Programm beendet und der Befehlszähler auf 00 gesetzt.	
14 15 16	106 700 606	Lädt den Inhalt von Adresse 06 in den Akkumulator, subtrahiert den Inhalt von Adresse 00 und speichert das Resultat unter Adresse 06.	
17 50 51	350 507 900	Testet, ob die Zahl im Akkumulator negativ ist. Falls ja wird der Inhalt von Adresse 07 ausgegeben, das Programm beendet und der Befehlszähler auf 00 gesetzt.	
18	870	Setzt den Befehlszähler auf 70. Mult. analog zu oben, kopiere die Faktoren in neue Adressen und setzte das Resultat der Mult. zu Beginn auf null. Statt das Resultat der Mult. auszugeben wird das Resultat der Mult. unter Adresse 07 gespeichert und dann mit 814 zurückgesprungen.	

Memory

00: 001	10: 106	20:	30:	40: 500	50: 507	60:	70: 107	80: 167	90: 167
01: 000	11: 700	21:	31:	41: 900	51: 900	61:	71: 665	81: 265	91: 607
02:	12: 606	22:	32:	42:	52:	62:	72: 105	82: 667	92: 814
03:	13: 340	23:	33:	43:	53:	63:	73: 666	83: 876	93:
04:	14: 106	24:	34:	44:	54:	64:	74: 101	84:	94:
05: 002	15: 700	25:	35:	45:	55:	65: 008	75: 667	85:	95:
06: -001	16: 606	26:	36:	46:	56:	66: -001	76: 166	86:	96:
07: 016	17: 350	27:	37:	47:	57:	67: 016	77: 700	87:	97:
08:	18: 870	28:	38:	48:	58:	68:	78: 666	88:	98:
09:	19:	29:	39:	49:	59:	69:	79: 390	89:	99: 893

Deck

Load

Reader

CPU

PC: 00

Instruction Decoder

Instruction Register: 900

Opcode: HRS Operand: 0

Accumulator: -0001

Reset Clear Mem

Step Slow Run Halt

Output

016