ti23 assignment 11 Alabrsh Panov Zeitler

1a) Function oft the given Program:
As suitable input we will Choose the Values: v = 2 (Adr 13) / x = 3 (Adr 15) / y=4 (Adr 12) / z = 5 (Adr 14)
1.Durchgang (run1):

#	label	comands	description
0	low:	LDA (13)	Wert 2(v) wird aus Adresse 13 des Speichers geladen und Wert **2**(v) wird als Accumulatorwert gesetzt.
1		ADD (15)	Addiere Wert 3(x) aus Adresse 15 des Speichers zu dem aktuellen Accumulatorwert → 2+3=5 → aktueller AccWert ist **5**.
2		BRC ub:	Wenn CarryFlag 1 ist, dann wird zu „ub“ gesprungen (Zeile6). Wenn CarryFlag 0 ist, dann geht es weiter bei Zeile 3. → CarryFlag ist hier **0**
3		STA 15	Aktueller AccWert = 5 wird in Adresse 15 des Speichers gespeichert →d.h. in Adresse 15 steht jetzt **5**
4		LDA #0	Wert 0 wird als aktueller AccWert gesetzt → aktueller AccWert = **0**
5		JMP high:	ProgramCounter = 8 → springt in Zeile 8
6	ub:	STA 15	Man würde in Zeile 6 springen wenn CarryFlag 1 ist, dann würde aktueller AccWert in Adresse 15 des Speichers gespeichert werden.
7		LDA #1	Wert 1 würde als aktueller AccWert gesetzt werden.
8	high:	ADD (12)	Addiere Wert 4(y) aus Adresse 12 des Speichers zum aktuellen AccWert → 0+4=4 → aktueller AccWert = **4**
9		ADD (14)	Addiere Wert 5(z) aus Adresse 14 des Speichers zum aktuellen AccWert → 4+5 = 9 → aktueller AccWert = **9**
10		STA 14	Aktueller AccWert = 9 wird in Adresse 14 des Speichers gespeichert →d.h. in Adresse 14 steht jetzt **9**
11	exit:	JMP exit:	ProgramCounter = 11 → springt in zeile 11 und exit

Zwischenstand: Adr 12 = 4 / Adr 13 = 2 / Adr 14 = 9 / Adr 15 = 5
2.Durchgang (run2):

#	label	comands	description
0	low:	LDA (13)	Wert 2 wird aus Adresse 13 des Speichers geladen und Wert **2** wird als Accumulatorwert gesetzt.
1		ADD (15)	Addiere Wert 5 aus Adresse 15 des Speichers zu dem aktuellen AccWert → 2+5=7 → aktueller AccWert ist **7**.
2		BRC ub:	Wenn CarryFlag 1 ist, dann wird zu „ub“ gesprungen (Zeile6). Wenn CarryFlag 0 ist, dann geht es weiter bei Zeile 3. → CarryFlag ist hier **0**
3		STA 15	Aktueller AccWert = 7 wird in Adresse 15 des Speichers gespeichert →d.h. in Adresse 15 steht jetzt **7**
4		LDA #0	Wert 0 wird als aktueller AccWert gesetzt → aktueller AccWert = **0**
5		JMP high:	ProgramCounter = 8 → springt in Zeile 8
6	ub:	STA 15	Man würde in Zeile 6 springen wenn CarryFlag 1 ist, dann würde aktueller AccWert in Adresse 15 des Speichers gespeichert werden.
7		LDA #1	Wert 1 würde als aktueller AccWert gesetzt werden.
8	high:	ADD (12)	Addiere Wert 4 aus Adresse 12 des Speichers zum aktuellen AccWert → 0+4=4 → aktueller AccWert = **4**
9		ADD (14)	Addiere Wert 9 aus Adresse 14 des Speichers zum aktuellen AccWert → 4+9 = 13 → aktueller AccWert = **13**
10		STA 14	Aktueller AccWert = 13 wird in Adresse 14 des Speichers gespeichert →d.h. in Adresse 14 steht jetzt **13**
11	exit:	JMP exit:	ProgramCounter = 11 → springt in zeile 11 und exit

Endergebnis: Adr 12 = 4 / Adr 13 = 2 / Adr 14 = 13 / Adr 15 = 7 / End AccWert = 13

Erleuterung Funktion des Programms:
Interpretiert man die Daten in [12] und [13] als eine einzige Zahl doppelter Größe (8 bit) und das gleiche für [14] und [15], dann führt das Programm die Addition beider Zahlen (in zwei 4-bit-Additionen aufgeteilt) aus und überschreibt die beiden rechten Register ([14] und [15]) mit dem Ergebnis.
Die obige Rechnung in Binärdarstellung:

run	cycle	[12]	[13]	[14]	[15]
1	0	0100	0010	0101	0011
	11	0100	0010	1001	0101
2	0	0100	0010	1001	0101
	11	0100	0010	1101	0111

```
  0100 0010
+ 0101 0011
```

```
  0100 0010
+ 1001 0101
```

1b) How to Improve the module computer to significantly simplify calculations of this kind ?

2a) Following Assembly Code in the language of our model computer determines the result oft he modulo operation x%y fort he two integers x and y stored in register (13) und (14).

Bsp.1: 7%4 = 3

Cycle	Zeile	label	command	NegFlag	[13]	[14]	[15]	AccWert
					7	4	0	
01	0		LDA (13)	0	7	4	0	7
02	1		STA 15	0	7	4	7	7
03	2		SUB (14)	0	7	4	7	3
04	3		BRN exit:	0	7	4	7	3
05	4		STA 15	0	7	4	3	3
06	5		JMP sub:	0	7	4	3	3
07	2	sub:	SUB (14)	1	7	4	3	-1
08	3		BRN exit:	1	7	4	3	-1
09	7	exit:	JMP exit:					

Bsp.2: 23%24 = 3

Cycle	Zeile	label	command	NegFlag	[13]	[14]	[15]	AccWert
					23	24	0	
01	0		LDA (13)	0	23	24	0	23
02	1		STA 15	0	23	24	23	23
03	2		SUB (14)	1	23	24	23	-1
04	3		BRN exit:	1	23	24	23	-1
05	7	exit:	JMP exit:					

2b) Used assembler program with the modulo calculation to translate the C implementation oft he GCD into a corresponding assembly program in the language of our simple module computer from the lecture.
Verified correctness of program by running it with 2 suitable pairs oft he value and documented it in report.

Aufgabenbearbeitung:

Aufgabe 1a,b→ Christian, Cora, Rahaf
Aufgabe 2a → Christian, Cora, Rahaf