Lernziele:

Die folgenden Konzepte sollen verstanden werden:
- Die sequentielle Ausführung eines Programms.
- Die Dekodierung und Ausführung einzelner Maschinenbefehle.
- Die Architektur der ULM (Ulm Lecture Machine).

Schritt 1: Darstellung von Unsigned- und Signed-Integer

Gib jeweils den Wert in Dezimaldarstellung an, der den Bitmustern in den folgenden Beispielen zugeordnet ist:

- u(0x012) =
- s(0x012) =
- s(0xFFA) =

Schritt 2: Erweiterung von Bitmustern

Setze jeweils auf der rechten Seite ein 64-stelliges Bitmuster in Hexadezimaldarstellung ein, sodass folgende Gleichungen gelten:

- u(0x012) = u(0x)
- s(0x012) = u(0x)
- u(0xFFA) = u(0x)
- s(0xFFA) = u(0x)

Schritt 3: Addition und Subtraktion

Setze jeweils auf der rechten Seite ein 64-stelliges Bitmuster in Hexadezimaldarstellung ein, sodass folgende Gleichungen gelten:

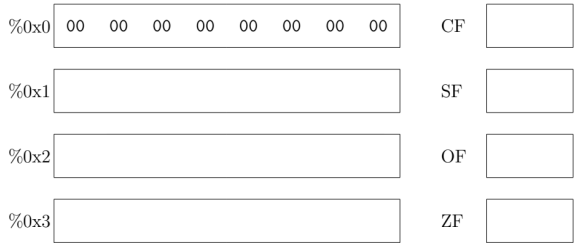
- (u(0x012) + u(0x01)) mod 2^{64} = u(0x)
- (u(0x012) - u(0x01)) mod 2^{64} = u(0x)
- (u(0x01) - u(0x012)) mod 2^{64} = u(0x)

Schritt 4: Dekodierung eines Befehls

Gegeben ist die folgende Beschreibung einer 32-bit Instruktion:

$u(\text{imm}) \mod 2^{64} \rightarrow u(\%\text{dest})$

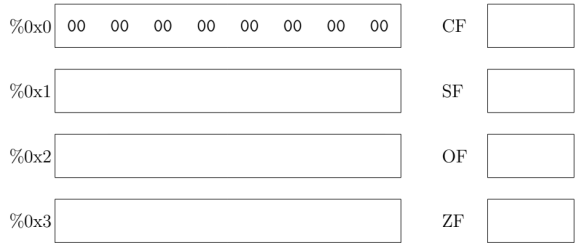
Beschreibe den Inhalt der Register %0, %1, %2, %3 nach Ausführung des Befehls mit dem Bitmuster 0x10100020. Nicht eingetragene Werte sollen als undefiniert betrachtet werden:



Schritt 5: Sonderfall für Register %0

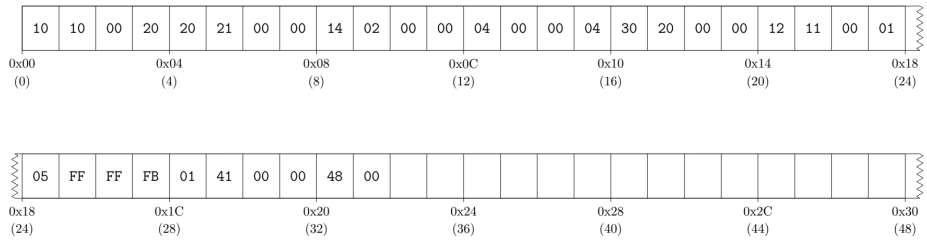
Das Register %0 hat die Besonderheit, dass es immer ein 64-stelliges Bitmuster hat, bei dem alle Bits Null sind. Wenn ein Bitmuster in Register %0 geschrieben wird, hat dies keinen Effekt. Beim Lesen von Register %0 erhält man immer ein Bitmuster ausschließlich mit Null-Werten.

Beschreibe den Inhalt von %0 nach Ausführung des Befehls 0x10000020:



Schritt 6: Beschreibung von Speicherinhalten

Das folgende Bild zeigt die ersten 48 Bytes eines Speichers. Jedes Kästchen stellt ein Byte (8 Bits) dar. Der Inhalt einer Speicherzelle wird mit jeweils zwei Hex-Ziffern dargestellt. Die Adressen der Speicherzellen sind sowohl in Hexadezimal- als auch in Dezimaldarstellung angegeben:



Eine Notation für eine formale Beschreibung des Inhalts einzelner Speicherzellen oder eines Speicherblocks soll anhand der folgenden zwei Spezialfälle eingeführt werden:

- M_1(0x8) steht für das Byte der Speicherzelle mit der Adresse u(0x8). In diesem Fall ist also M_1(0x8) = 0x14.
- M_4(0x8) steht für den Inhalt der vier aufeinanderfolgenden Speicherzellen, wobei u(0x8) die Adresse der ersten Speicherzelle ist. In diesem Fall ist M4(0x8) = 0x14020000.

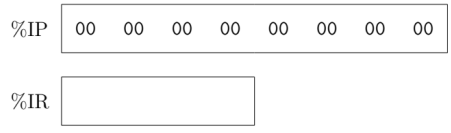
Allgemein beschreibt M_k(addr) einen Block mit k Speicherstellen, der bei Adresse addr beginnt.

Gib jeweils die folgenden Speicherinhalte an:

- M_4(0x010) =
- M_1(0x00020) =

Schritt 7: Register %IP und Register %IR

Das Register %IP (Instruction Pointer) ist ein 64-Bit-Register, und das Register %IR (Instruction Register) ist ein 32-Bit-Register. Der Inhalt dieser Register wird wie in diesem Bild beschrieben:



Beschreibe den Inhalt von %IP und %IR nach Ausführung eines Befehls mit dem Effekt

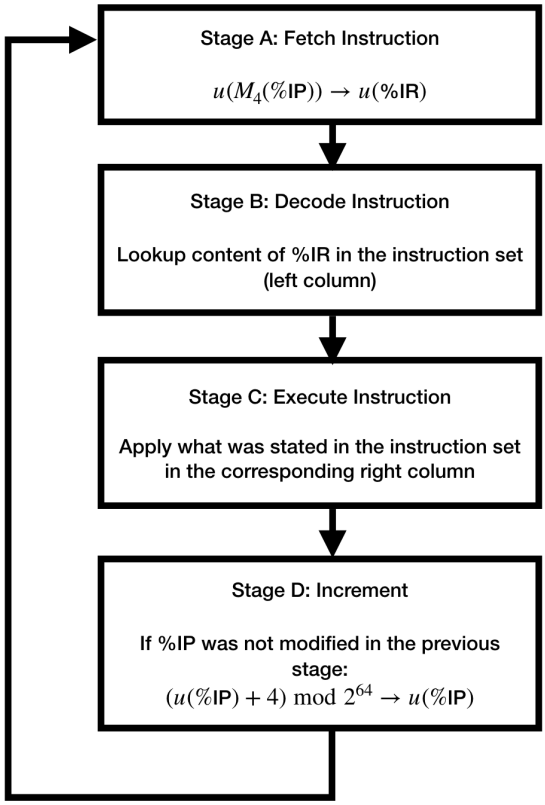
$$u(M_4(\%\text{IP})) \rightarrow u(\%\text{IR})$$

Der Speicherinhalt sei wie in Schritt 6 beschrieben.

Schritt 8 "ULM on Paper"

Auf einem separaten Blatt erhaltet ihr eine Beschreibung der ULM, nachdem ein Programm in den Speicher geladen wurde, aber die Ausführung noch nicht begonnen hat. Ihr erhaltet ebenfalls den für diese Aufgabe relevanten Ausschnitt eines Befehlssatzes sowie eine ASCII-Tabelle.

Folgendes Ablaufdiagramm beschreibt, wie das Programm sequenziell (in sogenannte "Von-Neumann-Zyklen") ausgeführt wird:



Die Ausführung beginnt mit "Stage A". Wenn ein Zyklus ("Stage A" bis "Stage B") vollständig durchlaufen ist, wurde ein Befehl ausgeführt.

Simuliert die Ausführung des Programms, bis die Ausführung in "Stage C" abbricht, wenn eine Halt-Instruktion ausgeführt wird.

Schritt 9

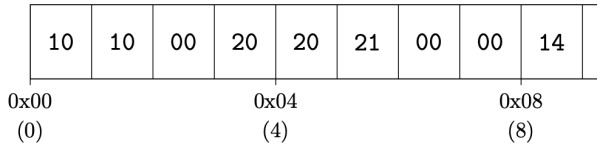
Schreibe ein Programm für die ULM, das den Text "Hallo Ulm!" ausgibt.

Memory

In drawings of the ULM the content of memory cells is represented in hexadecimal (skipping the '0x' prefix).

Let $A \in \{0,\ldots,2^{64}-1\}$ be an unsigned integer:
- $M_1(A)$ denotes the 8-bit pattern in memory cell with address A
- $M_4(A)$ denotes the 32-bit pattern of four consecutive memory cells where the first memory cell has address A.

Example: $M_1(5) = 0x21$, $M_4(4) = 0x20210000$



General Purpose Registers

This ULM variant has 16 registers denoted as %0x0, %0x1, …, %0xF:
- Each register has a width of 64 bits.
- Each register can be addressed with a 4-bit pattern (i.e. one hex digit).
- %0x0 is special. It always contains a bit pattern with only zeros. Writing to it has no effect.

Other Registers

- The instruction pointer (%IP) is 64 bits wide.
- The instruction register (%IR) is 32 bits wide.
- Each of the status flags ZF, CF, OF, SF can store a single bit.

Notation

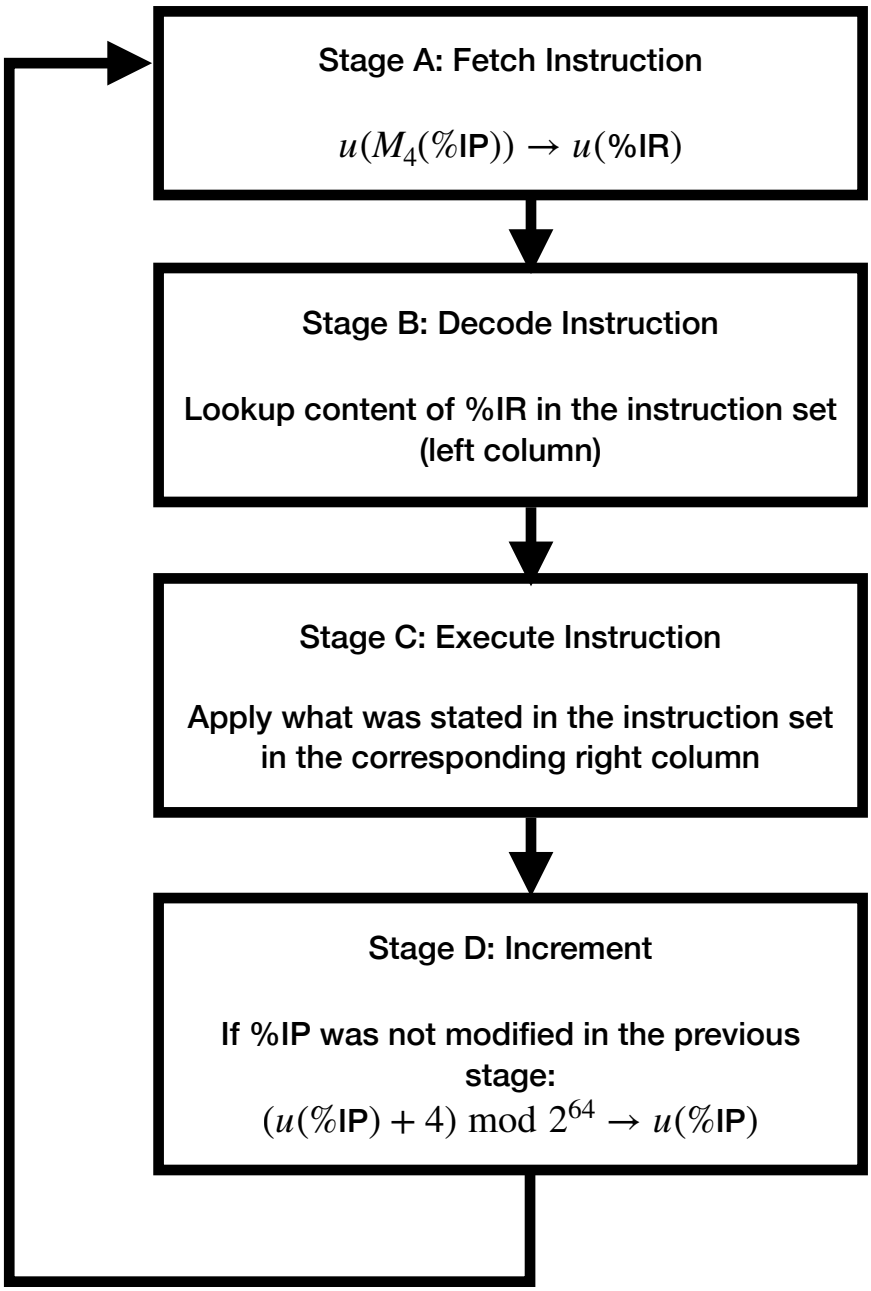
Let X be a bit pattern:
- $u(X)$ denotes the represented unsigned integer value
- $s(X)$ denotes the represented signed integer value

Let X and Y be a two bit pattern:
- $u(X) \rightarrow u(Y)$ denotes that Y gets modified such that $u(X) = u(Y)$. This is also called a "Zero extension of X".
- $s(X) \rightarrow s(Y)$ denotes that Y gets modified such that $s(X) = s(Y)$. This is also called a "Signed extension of X".

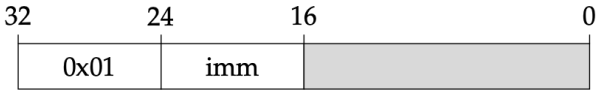
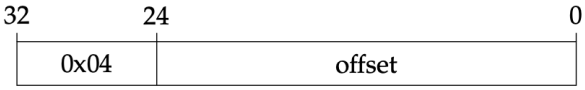
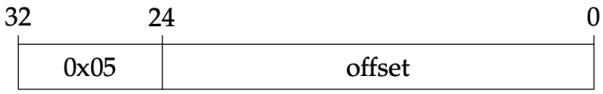
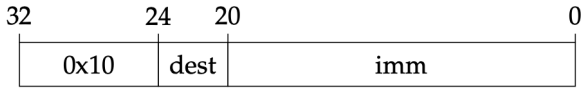
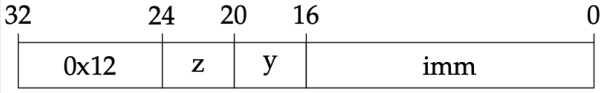
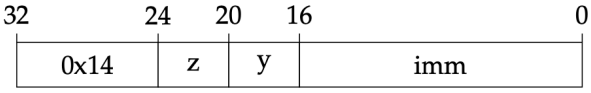
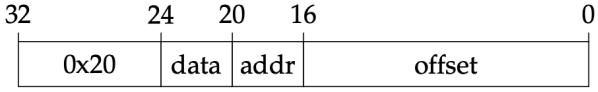
Example:
- $u(0x8F) \rightarrow u(\%0x1)$ writes $0x$000000000000008F to $\%\,0x1$
- $s(0x8F) \rightarrow s(\%0x1)$ writes $0x$FFFFFFFFFFFFFF8F to $\%\,0x1$

Von Neumann Cycle

After a program was loaded the cycle described below is applied until in stage C an halt instruction was executed. For educational purposes: After stage D the counter for the instruction cycle gets incremented.



Instruction Set (for decoding and executing instructions)

Content of $\%$IR (instruction register)	**Effect in execution stage**
	Halt program execution with exit code $u(\text{imm})$
	if ZF = 1 then $(u(\%\text{IP}) + 4 \cdot s(\text{offset})) \bmod 2^{64} \to u(\%\text{IP})$
	$(u(\%\text{IP}) + 4 \cdot s(\text{offset})) \bmod 2^{64} \to u(\%\text{IP})$
	$u(\text{imm}) \mod 2^{64} \to u(\%\text{dest})$
	$(u(\%\text{y}) + u(\text{imm})) \mod 2^{64} \to u(\%\text{z})$ Update status flags: Flag Condition ZF $u(\%y) + u(\%x) = 0$ CF $u(\%y) + u(\%x) \geq 2^{64}$ OF $s(\%y) + s(\%x) \notin \{-2^{63}, \ldots, 2^{63} - 1\}$ SF $s(\%y) + s(\%x) < 0$
	$(u(\%\text{y}) - u(\text{imm})) \mod 2^{64} \to u(\%\text{z})$ Update status flags: Flag Condition ZF $u(\%y) - u(imm) = 0$ CF $u(\%y) - u(imm) < 0$ OF $s(\%y) - s(imm) \notin \{-2^{63}, \ldots, 2^{63} - 1\}$ SF $s(\%y) - s(imm) < 0$
	$u(M_1(A)) \to u(\%\text{data})$ where $A = (u(\%\text{addr}) + s(\text{offset})) \bmod 2^{64}$
	Print the character with ASCII code $u(\%\text{x}) \bmod 2^8$

ASCII Table

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□