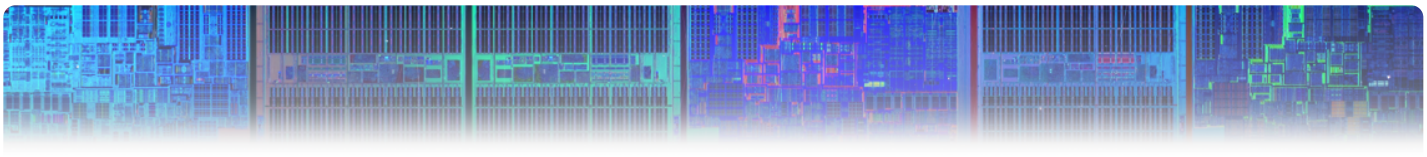

Zusatzaufgabe: GRAsm-Interpreter (Assembly)

Hinweis:
Falls Sie für das Praktikum 8 ECTS benötigen (siehe Studienplan), ist das Bestehen dieser Aufgabe zum Bestehen des Praktikums verpflichtend. Wenn dies nicht auf Sie zutrifft, können Sie die Aufgabe natürlich trotzdem bearbeiten.

In dieser Aufgabe soll ein Interpreter für *GRAsm* implementiert werden. Dieser soll ein in Bytecode vorliegendes Programm korrekt ausführen sowie möglicherweise auftretende Fehler erkennen. Als Abstraktion für die "GRA-Maschine" steht ein virtueller *State* zur Verfügung, auf dem die Operationen arbeiten sollen.

Zur Verfügung stehen die Folgenden "virtuellen" 64-bit Register:

Register	Spezielle Funktion
`ip`	Instruction Pointer
`ac`	Akkumulatorregister, Rückgaberegister
`r0-r7`	GP-Register

Diese sind als `struct grasm_state { uint64_t ip, acc, r0, .., r7; };` im Speicher angeordnet.

In der folgenden Tabelle finden Sie die verfügbaren Instruktionen sowie Encodings.

Instruktionstabelle ausklappen

Instruktion	Immediate	Encoding	Funktion
`stop`	--	`0x01`	Beendet die Ausführung
`nop`	--	`0x0f`	Keine Funktion
`set`	`uint64_t`	`0x10 <imm>`	`ac = imm`
`set rX`	`uint64_t`	`0x18 <0X> <imm>`	`rX = imm`
`cpy rX`	--	`0x11 <0X>`	`ac = rX`
`cpy rX rY`	--	`0x19 <XY>`	`rX = rY`
`add`	`uint64_t`	`0x20 <imm>`	`ac += imm`
`add rX`	`uint64_t`	`0x28 <0X> <imm>`	`rX += imm`
`add rX`	--	`0x21 <0X>`	`ac += rX`
`add rX rY`	--	`0x29 <XY>`	`rX += rY`
`sub`	`uint64_t`	`0x22 <imm>`	`ac -= imm`

Instruktion	Immediate	Encoding	Funktion
sub rX	uint64_t	0x2a <0X> <imm>	rX -= imm
sub rX	--	0x23 <0X>	ac -= rX
sub rX rY	--	0x2b <XY>	rX -= rY
mul	uint64_t	0x24 <imm>	ac *= imm
mul rX	uint64_t	0x2c <0X> <imm>	rX *= imm
mul rX	--	0x25 <0X>	ac *= rX
mul rX rY	--	0x2d <XY>	rX *= rY
xchg rX	--	0x26 <0X>	tmp = ac; ac = rX; rX = tmp
xchg rX rY	--	0x2e <XY>	tmp = rX; rX = rY; rY = tmp
and	uint64_t	0x30 <imm>	ac &= imm
and rX	uint64_t	0x31 <0X> <imm>	rX &= imm
and rX	--	0x32 <0X>	ac &= rX
and rX rY	--	0x33 <XY>	rX &= rY
or	uint64_t	0x34 <imm>	ac |= imm
or rX	uint64_t	0x35 <0X> <imm>	rX |= imm
or rX	--	0x36 <0X>	ac |= rX
or rX rY	--	0x37 <XY>	rX |= rY
xor	uint64_t	0x38 <imm>	ac ^= imm
xor rX	uint64_t	0x39 <0X> <imm>	rX ^= imm
xor rX	--	0x3a <0X>	ac ^= rX
xor rX rY	--	0x3b <XY>	rX ^= rY
not	--	0x3c	ac = ~ac
not rX	--	0x3d <0X>	ac = ~rX
not rX rY	--	0x3e <XY>	rX = ~rY
cmp rX	uint64_t	0x40 <0X> <imm>	ac = rX - imm
cmp rX rY	--	0x41 <XY>	ac = rX - rY
tst rX	uint64_t	0x42 <0X> <imm>	ac = rX & imm
tst rX rY	--	0x43 <XY>	ac = rX & rY
shr	uint8_t	0x50 <imm>	ac >>= imm
shr rX	uint8_t	0x51 <0X> <imm>	rX >>= imm
shr rX	--	0x52 <0X>	ac >>= rX
shr rX rY	--	0x53 <XY>	rX >>= rY
shl	uint8_t	0x54 <imm>	ac <<= imm

Instruktion	Immediate	Encoding	Funktion
`shl rX`	`uint8_t`	`0x55 <0X> <imm>`	`rX <<= imm`
`shl rX`	--	`0x56 <0X>`	`ac <<= rX`
`shl rX rY`	--	`0x57 <XY>`	`rX <<= rY`
`ld`	`uintptr_t`	`0x60 <imm>`	`ac = [imm]`
`ld rX`	`uintptr_t`	`0x61 <0X> <imm>`	`rX = [imm]`
`ld rX`	--	`0x62 <0X>`	`ac = [rX]`
`ld rX rY`	--	`0x63 <XY>`	`rX = [rY]`
`st`	`uintptr_t`	`0x64 <imm>`	`[imm] = ac`
`st rX`	`uintptr_t`	`0x65 <0X> <imm>`	`[imm] = rX`
`st rX`	--	`0x66 <0X>`	`[rX] = ac`
`st rX rY`	--	`0x67 <XY>`	`[rX] = rY`
`go`	`uintptr_t`	`0x70 <imm>`	`ip = <imm>`
`go rX`	--	`0x71 <0X>`	`ip = rX`
`gr`	`int16_t`	`0x72 <imm>`	`ip = ip + <imm>`
`jz`	`uintptr_t`	`0x73 <imm>`	`ip = ac == 0 ? <imm> : ip + sizeof(jz)`
`jz rX`	--	`0x74 <0X>`	`ip = ac == 0 ? rX : ip + sizeof(jz)`
`jrz`	`int16_t`	`0x75 <imm>`	`ip = ac == 0 ? ip + <imm> : ip + sizeof(jrz)`
`ecall`	`uintptr_t`	`0x80 <imm>`	`ac = <imm>(r0, .., r5)`
`ecall rX`	--	`0x81 <0X>`	`ac = rX(r0, .., r5)`

- Mit der Instruktion `ecall` werden (unbekannte) externe Funktionen an der gegebenen Adresse aufgerufen:
  - Achten Sie darauf die Calling Convention einzuhalten!
  - Die Funktionen nehmen maximal 6 Parameter entgegen, das Ergebnis soll in `ac` abgelegt werden.
- Soweit nicht anders angegeben werden alle Immediates im Little-Endian Format encoded.
  - d.h. `add 0x01234567 -> 20 67 45 23 01 00 00 00 00` .
- Achten Sie vor allem bei den jumps darauf, den `ip` richtig zu setzen!
  - `ip` wird am *Ende* einer (gültigen) Instruktion aktualisiert, d.h. *nach* etwaiger Fehlerbehandlung.
  - Sprünge setzen den `ip` explizit.
  - Relative Sprünge werden relativ zur *aktuellen* Instruktion berechnet.
- Die übergebenen Speicheradressen für `ld` / `st` und `ecall` sind gültig und referenzieren direkt den unterliegenden Speicher.
- Die oberen 4 Bits von `<0X>` sind für diese Aufgabe *dont-care*, sollen also nicht geprüft werden.
- Shifts können maximal 63 bits shiften, d.h. nur 6 Bits des Operanden werden betrachtet.

Bei möglicherweise auftretenden Fehlern sollen die folgenden Fehlercodes zurückgegeben werden:

Fehler	Fehlercode	Grund
Kein Fehler	`0`	--
Unbekannter Opcode	`-1`	Instruktion existiert nicht oder unvollständig
Out-of-Bounds-Zugriff	`-2`	`ip >= len` , Register nicht gültig

Bei mehreren "gleichzeitig" auftretenden Fehlern hat `-1` Präzedenz vor `-2` .

Beispielprogramm (Zeilenumbrüche, Offsets und Kommentare nur zur Visualisierung):

```
# ac = fac(r0) if r0 else 0   .   .   .
00: 19 10                     # cpy r1 r0
02: 40 01 00 00 00 00 00 00 00 # cmp r1 0
0c: 75 1f 00                  # jrz +31
0f: 40 01 01 00 00 00 00 00 00 # cmp r1 1
19: 75 12 00                  # jrz +18
1c: 2a 01 01 00 00 00 00 00 00 # sub r1 1
26: 2d 01                     # mul r0 r1
28: 72 e7 ff                  # gr  -25
2b: 11 00                     # cpy r0
2d: 01                        # stop
```

Implementieren Sie in x86-64 Assembly einen Interpreter, der die oben genannten Anforderungen erfüllt; insbesondere müssen *alle* Instruktionen implementiert werden.

Hinweis zur Bewertung:
Abweichend von den Hausaufgaben wird bei dieser Aufgabe ausschließlich die letzte Abgabe zur Bewertung herangezogen.

Signatur: `uint64_t grasm_interpreter(struct grasm_state* state, size_t len, uint8_t prog[len]);`

Für diese Aufgabe gibt es keinen Score. Sobald alle Tests bestanden wurden, erhält Ihre Abgabe den Status "Gelöst".

Deadline: 2023-06-11 23:59:00

Aktuelle Abgabe (2023-06-01 01:08:25) **Gelöst**

```
 1 .intel_syntax noprefix
 2 .global grasm_interpreter
 3 .text
 4
 5 //Signature: uint64_t grasm_interpreter(struct grasm_state* state, size_t len,
 6 //struct grasm_state { uint64_t ip, acc, r0, r1, r2, r3, r4, r5, r6, r7; }; 64
 7 //rdi: state
 8 //rsi: len
 9 //rdx: prog address
10
11 //All instructions ---> prog
12 //prog contains e.g. 0x01 (00 00 00 01) for stop
13
14 //The number of instructions ---> len
15 //state is just the current state of the program/system
16
17 //example: add 0x01234567 ->    20        67     45 23 01 00 00 00 00 ---> is
18 //                           00010000  01100111 ......
```