Lösungsvorschlag: Session 2

Teil 1:

Preprocessing: pp-debounce.S

PO.H = HI(FIO_FLAG_C)
wird zu
PO.H = ((0xFFC00704) >> 16) & 0xFFFF)

Die Adresse 0xFFC00704 wird um 16 Bits nach rechts geshiftet und mit lauter 1-en verundet, so erhält man den Wert 0xFFC0 was den ersten 16 Bits (ab MSB) entspricht. Diese Operation wird durch das HI(...) hervorgerufen.

Assembler: ~~debounce.S~~ debounce.o

Sourcecode pp-debounce.S $\xleftrightarrow[\text{bijektiv}]{1:1}$ Binärcode debounce.o (nicht ausführbar und nicht lesbar)

Linker:

debounce.o $\xrightarrow{\text{Linker}}$ debounce.x (ausführbar) nicht lesbar

Infos für Speicher, shared libraries

debounce.x $\xrightarrow{\text{ihex}}$ debounce.hex (ausführbar und "lesbar")

Ausführbarer Code:
Anfang: 4: "...6701 48E1..." NOP wurde ignoriert
Ende: 12: "...BD051000..."

Selbstkontrolle:

Grenzen: Je nach Komplexität kann der Code sehr unleserlich sein.

Teil 2

JUMP: Springt an eine bestimmte Adresse und führt die folgenden Operationen aus. Alle Register bleiben unverändert.

CALL: Ähnlich wie JUMP, mit dem Unterschied, dass die Adresse der nächsten Instruktion (vor dem Sprung) in das Register RETS gespeichert wird. Mit der Instruktion RTS (Return from Subroutine) springt man wieder zurück (Adresse: RETS)

LINK X: Speichert das Register RETS ~~auf d~~ und FP (Frame Pointer) auf den Stack,* und alloziiert zusätzlich X Bytes auf dem Stack. X muss ein Vielfaches von 4 sein.

Die Instruktion kann auch folgendermassen aufgefasst werden:

```
[--SP] = RETS;
[--SP] = FP;
FP = SP;
SP += -X;
```

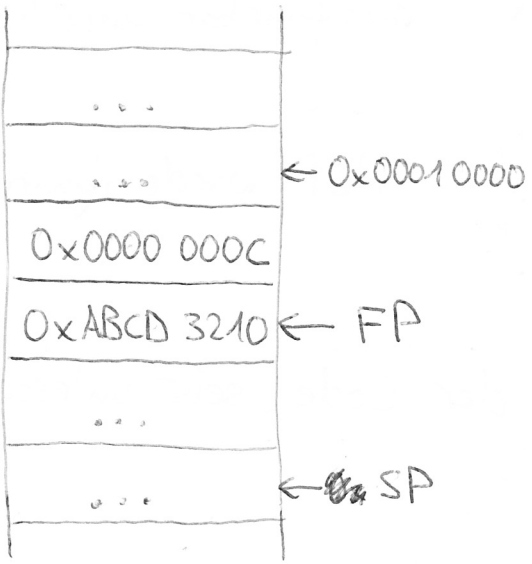
* setzt den aktuellen FP auf den Wert des Stackpointers

UNLINK:
```
SP = FP;
FP = [SP++];
RETS = [SP++];
```

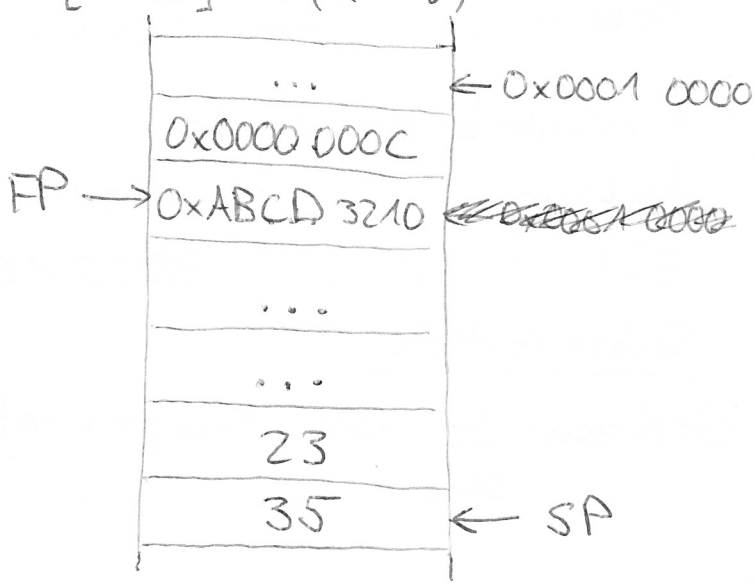
UNLINK und LINK gehören immer zusammen!

Anfang: PC = 0x0000 0000 FP = 0xABCD 3210

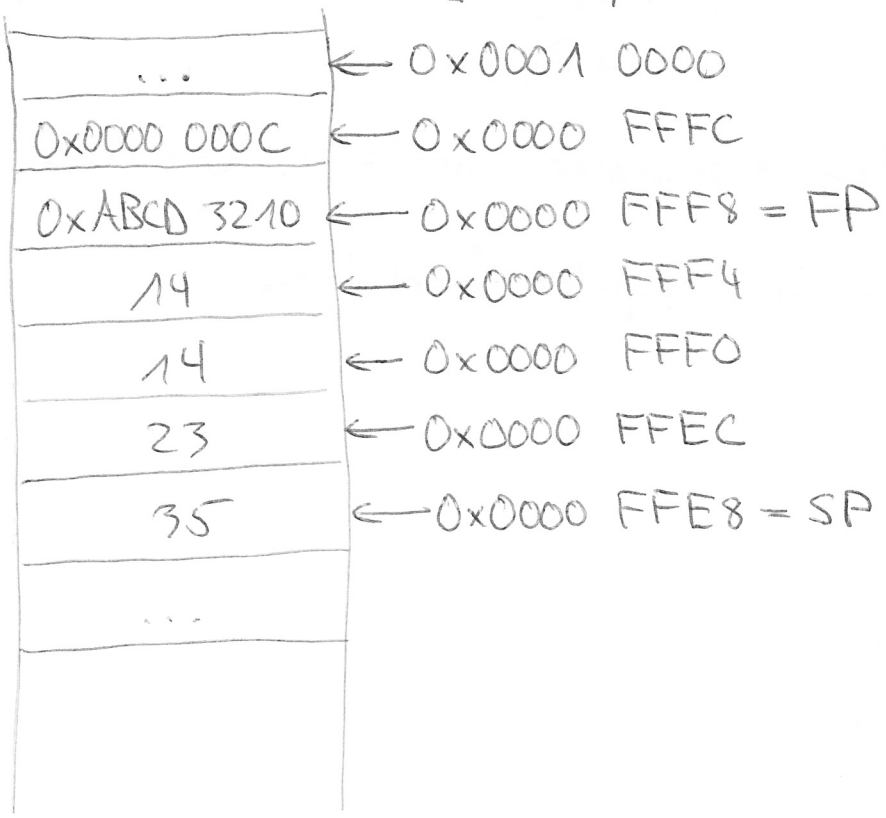
label_a: LINK 8



[--SP] = (R7:6)

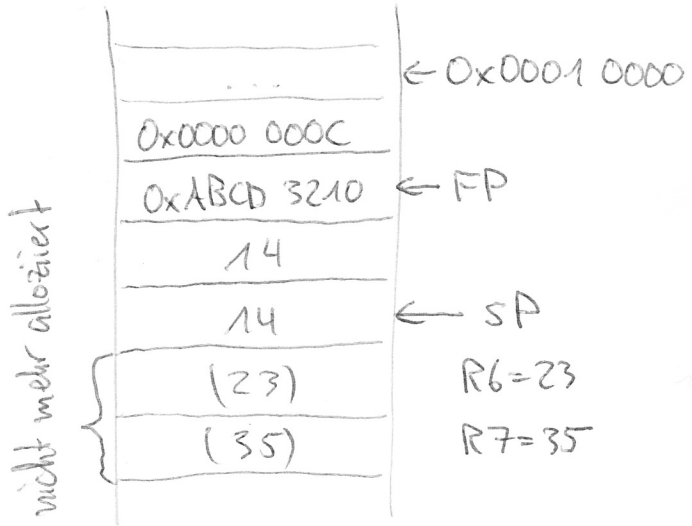


label_b: [FP-8] = R7;

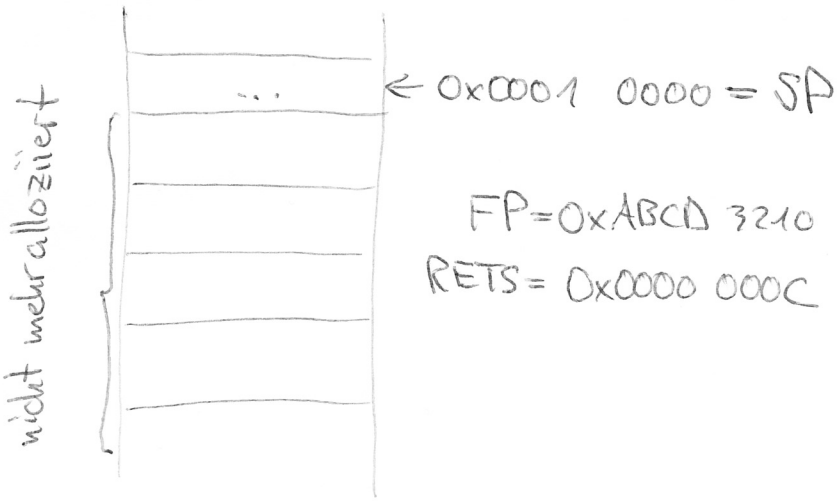


RETS = 0x0000 000C
R6 = 23
R7 = 14

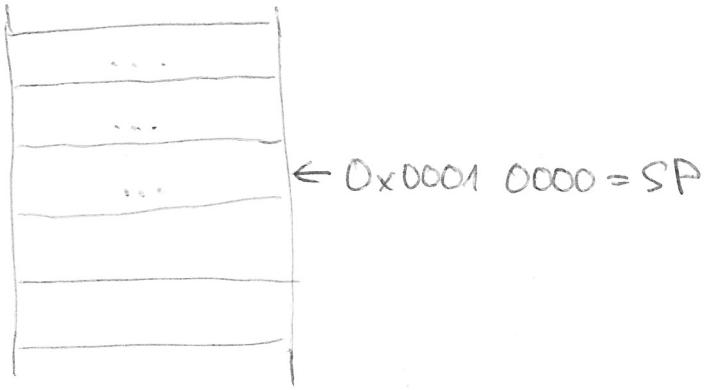
(R7:6) = [SP++]



UNLINK



label_c:



RETS = 0x0000 000C
FP = 0xABCD 3210
R6 = 23
R7 = 35

Bonus:

	vorher	nachher
Dateigrösse:	9.0kB	444 Bytes

Wir müssten den Code vor den Assembler ansetzen. Die Übersetzung würde durch einen Compiler erfolgen.