TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Technische Informatik, Professur für Prozessorentwurf

Rechnerarchitektur: Übungssatz 10

Aufgabe 10.1

Gegeben sei folgender Speicherauszug aus einem byteweise addressierbaren Speicher:

Adresse	Inhalt
0x0FD	0x02
0x0FE	0xD5
0x0FF	0x68
0x100	0x4F
0x101	0x73
0x102	0xC9
0x103	0x1B

(a) Welche Werte werden bei folgenden Zugriffen von der Speicheradresse 0x100 gelesen?

Endian	Byte	16-Bit-Wort	32-Bit-Wort
Little			
Big			

(b) An welcher Adresse befindet sich, je nach Endian, das niederwertigste Byte des 32-Bit-Wortes an Adresse 0x100?

Aufgabe 10.2

Ein Record bestehe aus mehreren Datenfeldern für Ganzzahlen unterschiedlicher Bitbreite:

```
record
  a : int16;      // 16 Bit
  b : int8;       //  8 Bit
  c : int16;      // 16 Bit
  d : int32;      // 32 Bit
end record;
```

In einem Programm werde nun eine Instanz dieses Records ab der Adresse 0xA38 im byteadressierten Speicher mit {.a=0x341A,.b=0x4D,.c=0x90F2,.d=0xA37B 672E} initialisiert. Zeigen Sie den Auszug des belegten Speichers nach dessen Initialisierung, wenn:

(a) das Record dicht in Little-Endian gepackt wird, und

(b) jedes Datenfeld des Records (ohne Umordnung!) gemäß seiner Größe *aligned* (= *natürliches Alignment*) in Big-Endian abgelegt wird!

Aufgabe 10.3

Ein Stack (Stapelspeicher, Kellerspeicher) wird oft mit Hilfe eines durch eine Speicheradresse repräsentierten Stackzeigers (SP) realisiert.

(a) Welche beiden naheliegenden alternativen Speicherplätze kann der Stackzeiger identifizieren?

(b) Wie ist der Stackzeiger bei einer Stackoperation (PUSH oder POP) zu korrigieren?

SP-**Ziel** (wie oben): **Wachstumsrichtung**			
aufwärts	PUSH POP		
abwärts	PUSH POP		

Aufgabe 10.4

Einem Stack begegnet man häufig als Aufrufstack, auf dem die aktuelle Hierarchie von Unterprogrammaufrufen in Form von Rücksprungadressen protokolliert ist.

(a) Wie lässt sich auf dem Stack weiterer Speicher für lokale Variablen reservieren?

(b) Wie und über welchen Adressierungsmodus kann auf so allokierte Variablen zugegriffen werden?

(c) Wie werden diese Variablen wieder freigegeben?

(d) Für welches Aufrufmuster ist diese Form der Speicherallokation im Gegensatz zu einer statischen Allokation an festen Speicheradressen zwingend erforderlich?

Aufgabe 10.5

Auf einem nach unten wachsenden Stack ist Speicherplatz für lokale Variablen reserviert:

	⋮
SP+3:	*Rücksprung-Adresse*
SP+2:	*a*
SP+1:	*b*
SP→	*c*

Lokale Variablen können durch `PUSH`-Befehle als Kopie auf den Stack gelegt werden bzw. durch `POP`-Befehle durch das obere Stackelement ersetzt werden:

`PUSH [SP+i]:` $v \leftarrow$ `<SP+i>`; `SP` $\leftarrow$ `SP - 1`; `<SP>` $\leftarrow v$
`POP [SP+i]:` $v \leftarrow$ `<SP>`; `SP` $\leftarrow$ `SP` $+ 1$; `<SP+i>` $\leftarrow v$

Die Parameterübergabe und Ergebnisrückgabe eines Funktionsaufrufes erfolgen über den Stack. Es soll nun die Funktion `f` mit den Parametern *a* und *b* aufgerufen und deren Ergebnis in *c* abgelegt werden.

(a) Mit welcher Codesequenz werden die Argumente für den Funktionsaufruf vorbereitet?

(b) Wie sieht der Stack am Anfang der Funktion `f` direkt nach deren Aufruf aus? Speicherplatz für etwaige lokale Variablen wurde noch nicht reserviert.

(c) Welche Codesequenz speichert das auf dem Stack zurückgegebene Ergebnis nach *c*?

(d) Welche Adressierungsmodi finden sich in dem Befehl `PUSH [SP+2]`?

(e) Wie lässt sich die Abhängigkeit des Variablenoffsets vom sich regelmäßig ändernden Stackzeiger vermeiden?

Aufgabe 10.6

Von einer 8-Bit-Stackarchitektur mit 12-Bit-Adressraum seien folgende Befehle bekannt:

Mnemonik	Maschinencode	Beschreibung
`NOP`	0000 0000	Tue nichts.
`POP`	0001 0000	Verwirf oberstes Stackelement.
`AND`	0001 0001	Nimm oberen beiden Werte vom Stack, UND-verknüpfe sie bitweise und lege das Ergebnis wieder auf den Stack.
`OR`	0001 0010	Nimm oberen beiden Werte vom Stack, ODER-verknüpfe sie bitweise und lege das Ergebnis wieder auf den Stack.
`XOR`	0001 0011	Nimm oberen beiden Werte vom Stack, XOR-verknüpfe sie bitweise und lege das Ergebnis wieder auf den Stack.
`ADD`	0001 0100	Nimm oberen beiden Werte vom Stack, addiere sie und lege das Ergebnis wieder auf den Stack.
`PUSH #i`	1100 `<i>`	Lege vorzeichenerweiterte 4-Bit-Konstante *i* auf den Stack.
`DUP`	1101 1101	Dupliziere obersten Stackwert noch einmal auf den Stack.
`PUSH [A]`	1110 `<A.hi>, <A.lo>`	Kopiere Speicherinhalt von Adresse *A* auf den Stack.
`POP [A]`	1111 `<A.hi>, <A.lo>`	Verschiebe obersten Stackwert an die Speicheradresse *A*.

(a) Wie lassen sich das Einer- bzw. das Zweierkomplement des obersten Stackelementes *x* bilden? Geben Sie die erforderlichen Befehlssequenzen in Mnemoniken und im hexadezimalen Maschinencode an! Dokumentieren Sie den Stackzustand!

(b) An den Speicheradressen `0x230` ... `0x232` liegen die Werte der Variablen *a*, *b* und *c* in dieser Reihenfolge. Geben Sie wie oben ein Programm zur Berechnung von $y = 3(a - b) - c$ an! Das Ergebnis *y* soll abschließend an der Adresse `0x240` abgelegt werden.