
1. Aufgabenblatt mit Lösungen

Problem 1: Begriffe (je 1 = 9)

Beschreiben Sie jeden der folgenden Begriffe durch maximal 2 Sätze:

- a) Difference Engine
- b) Relais
- c) EPROM
- d) Core (wie in Quad-Core-CPU)
- e) ALU
- f) ISA
- g) Benchmark
- h) Turing-Maschine
- i) MSIL/CIL

Solution 1

- a) Die **Difference Engine** ist eine von Charles Babbage entwickelte Maschine. Das erste Modell wurde ab 1821 gebaut. Sie wurde konzipiert um Addition und Subtraktion auf Polynomen durchzuführen und stetig weiterentwickelt. 1847 wurde die Difference Engine 2 konstruiert. Diese hatte ein ähnliches Design wie die Analytical Engine und benötigte etwa ein Drittel der Teile ihres Vorgängermodells.
- b) Das **Relais** ist der Vorgänger des Transistors. Es besteht aus einer Spule und einem Schalter. Falls Strom durch die Spule fließt, zieht das dann entstehende Magnetfeld den Schalter in die schließende oder öffnende Position. Relais fanden u.a. bis zur Entwicklung von Transistoren Verwendung in Computern.
- c) Das **EPROM** ist ein nicht flüchtiger Nur-Lese-Speicher, der bei Bedarf zum Beispiel mit UV-Strahlung (ältere Modelle) gelöscht werden kann. In vielen Fällen ist die Anzahl der Löschmöglichkeiten begrenzt.
- d) Core bezeichnet im Allgemeinen den Prozessorkern eines Computers. Bei Ein-Kern-Prozessoren gibt es genau 1 CPU und es ist keine echte Parallelität im Prozess- oder Programm-Level möglich. Mehr-Kern-Prozessoren haben mehrere CPU und können so echt parallel verschiedene Programme ausführen.
- e) Die **ALU** ist die Arithmetisch-Logische Einheit einer CPU und für die Berechnung verantwortlich. Dabei werden die Operanden von der Steuereinheit an die ALU gesandt und die Ergebnisse auch wieder abgeholt. Es gibt häufig mehrere ALU, die entweder die gleichen arithmetischen Funktionen zur Verfügung stellen oder aber eine zusätzliche Funktionen haben, wie z.B. Fließkomma- oder Vektorarithmetik.

TI II

Sommersemester 2010

PD Dr. Katinka Wolter

- f) **ISA** steht für Instruction Set Architecture oder auch Industry Standard Architecture. Im ersten Fall handelt es sich um Level 2 des Computerschichtenmodells und beschreibt die dem Prozessor zur Verfügung stehenden Assembler-Befehlssatz. Im zweiten Fall steht die Abkürzung für ein 16-Bit Bussystem, welches auf Grund seiner geringen Geschwindigkeit (8 MHz) heutzutage kaum noch Verwendung findet.
- g) **Benchmark** Ein Benchmark ist eine Zusammenfassung mehrerer Testroutinen oder -programme, die hinsichtlich der vorkommenden Befehle oder Befehlskombinationen wohl definiert sind und sind z.B. auf spezielle Eigenschaften einer CPU eines Computers zugeschnitten. Die Durchführung eines solchen Benchmark gibt ein Testergebnis zurück, welches die Leistungsfähigkeit des getesteten Computers bezüglich eines Einheitscomputers bewertet.
- h) **Turing-Maschine** Eine Turingmaschine ist ein theoretisches Modell eines Computers. Hauptmerkmal ist, dass sie unbegrenzte Speicherplätze hat und jeder Speicherplatz ein beliebiges Datum aufnehmen kann. Die Anzahl der Zustände ist endlich. Ebenso ist genau definiert, dass zu jedem Zustand genau ein Folgezustand existiert.
- i) **MSIL/CIL** (Microsoft) Common Intermediate Language ist ein Zwischencode des Assembly language Level (Level 4). Dieser wird aus der höheren Programmiersprache erzeugt und erst zur Laufzeit von einem JIT-Compiler in Maschinencode übersetzt.

Problem 2: Das von-Neumann-Rechnermodell (je 1 = 10)

1946 wurde das von-Neumann-Rechnermodell vorgestellt, das die Rechnerarchitektur bis heute maßgeblich beeinflusst. Arbeiten Sie die grundlegenden Organisationsprinzipien und Besonderheiten dieses Modells (ggf. anhand des siebten Kapitels (im Lehrbuch von Oberschelp/Vossen)) heraus, indem Sie folgende Fragen möglichst prägnant und in eigenen Worten beantworten.

- a) Mit dem von-Neumann-Rechnermodell wurde erstmalig das Konzept für einen echten „general-purpose Computer“ vorgeschlagen. Was ist darunter zu verstehen?
- b) „Programme sind auch nur Daten“ ist eine grundlegende und eng mit dem von-Neumann-Rechnermodell verbundene Sichtweise. Was ist darunter zu verstehen?
- c) Das von-Neumann-Rechnermodell setzt sich aus drei Hauptbestandteilen zusammen. Welche Bestandteile sind dies und welchem Zweck dienen sie?
- d) Im von-Neumann-Rechnermodell ist der Datenprozessor ein Bestandteil der CPU. Welche Aufgaben werden von welchen Komponenten dieses Prozessors erfüllt?
- e) Im von-Neumann-Rechnermodell ist der Befehlsprozessor ein Bestandteil der CPU. Welche Aufgaben werden von welchen Komponenten dieses Prozessors erfüllt?
- f) Das von-Neumann-Rechnermodell unterscheidet zwischen Daten- und Adressbus. Warum macht das Sinn? Es ergeben sich auch Zusammenhänge zwischen der Größe (in Bits) des MAR, des MBR, des Speichers, einer Speicherzelle sowie der Speicherzellenanzahl. Welche?
- g) Die Arbeitsweise eines von-Neumann-Rechners wird durch die Bezeichnung SISD allgemein charakterisiert. Welches Prinzip verbirgt sich hinter dieser Abkürzung?
- h) Bahnbrechend neu am von-Neumann-Rechnermodell war das Konzept einer quasi universellen Programmierbarkeit. Erörtern Sie in diesem Zusammenhang die Begriffe Maschinencode, Assemblersprachen sowie Ein- und Mehr-Adress-Befehle.
- i) Charakteristisch für das von-Neumann-Rechnermodell ist ein Zwei-Phasen-Konzept der Befehlsverarbeitung. Welches Problem wird damit auf welche Weise gelöst?

TI II

Sommersemester 2010

PD Dr. Katinka Wolter

- j) Die Architektur eines klassischen von-Neumann-Rechners führte schon bald zu einem gewichtigen Problem, dem von-Neumannschen „Flaschenhals“. Was ist darunter zu verstehen und wie versuchte man später dieses Problem zunächst zu umgehen?

Solution 2

- a) Die Struktur dieses Rechners ist im Gegensatz zu den endlichen Automaten vollkommen unabhängig von einem speziellen Problem, da man für jedes neue Problem ein eigenes Programm im Speicher ablegt.
- b) Programme werden als Bitfolge in dem selben Speicher abgelegt wie auch die Daten selber, daher besteht rein praktisch kein Unterschied zwischen Programmen und Daten. (Daten und Programme sind nur dadurch zu unterscheiden, wie beim Programmablauf auf die Bitfolgen zugegriffen wird.)
- c) Ein von-Neumann-Rechner besteht aus:
- Zentraleinheit (CPU): Ablaufsteuerung und Ausführung von Befehlen
 - Speicher: Enthält Daten und Programme als Bitfolge
 - Ein/Ausgabe-Einheit: Verbindung zur Außenwelt
- d) Der Datenprozessor besteht aus:
- ALU (Arithmetic Logical Unit): Rechenwerk zur Ausführung von Berechnungen
 - Akku (Akkumulator): „General-purpose-“Register, kann für jede im Rahmen eines Programms anfallende Aufgabe verwendet werden
 - MR (Multiplikator Register): z.B. Aufnahme von Multiplikationsergebnissen
 - L (Link Register): Aufnahme eines Additionsübertrags
 - MBR (Memory Buffer Register): Wickelt die Kommunikation mit dem Speicher ab
- e) Der Befehlsprozessor besteht aus:
- IR (Instruction Register): Enthält den aktuell bearbeiteten Befehl
 - MAR (Memory Adress Register): Enthält die Adresse des Speicherplatzes, der als nächstes anzusprechen ist
 - PC (Program Counter): Enthält die Adresse des nächsten Befehls
 - Decodierer: Entschlüsselt Befehle
 - Steuerwerk: Steuert die Ausführung
- f) Es macht Sinn, da die Länge einer Speicherzelle im Allgemeinen verschieden ist von der Länge ihrer Adresse:
- Falls eine Speicherzelle m Bits aufnehmen kann, zur Darstellung einer Adresse n Bits erforderlich sind und $m \neq n$ ist, so ist es nicht sinnvoll, den gleichen Bus zur Übertragung von Adressen und Daten zu benutzen, da z.B. im Falle $m > n$ ein m -Bit-Bus mit der Übertragung einer Adresse nicht ausgelastet wäre.
- Zusammenhang der Größen:
- Besitzt ein Speicher 2^n Plätze, so benötigt man n Bits um auf eine Speicherzelle zuzugreifen. Der Adressbus muss also in der Lage sein n Bits in einem Takt zu transportieren.
- Wenn ein Rechner einen n -Bit Datenbus hat, so bedeutet dies, dass er höchstens n Bits in einem Zugriff ansprechen kann.
- Hieraus ergibt sich, dass (bei getrenntem Daten- und Adressbus) die breite des Datenbusses mit der Länge des MBR und die des Adressbusses mit der Länge des MAR übereinstimmen muss.

TI II

Sommersemester 2010

PD Dr. Katinka Wolter

- g) SISD - Single Instruction Single Data: Zu jedem Zeitpunkt führt die CPU genau einen Befehl aus, und dieser kann (höchstens) einen Datenwert bearbeiten.
- h) Eine Befehlsabfolge ist eine Folge von Binärzahlen, die nach Maschinencode aufgebaut ist. Da Befehle in dieser Form schwer lesbar sind stellt man dem Benutzer lesbarere Darstellungen zur Verfügung, wie z.B. Assemblersprachen die zu jedem Befehl einen speziellen Mnemocode bereithalten. Da der Akku bei jeder arithmetischen oder logischen Operation beteiligt ist, muss man ihn meist nicht explizit in den Befehlen angeben.
- Einstellige Operationen (Negation, Wurzelziehen,...) kommen somit ohne Operanden aus. Zweistellige Operationen (Addition, Multiplikation,...) kommen mit einem Operanden aus, welcher dann mit dem Inhalt des Akkus verknüpft wird. Dies nennt man Ein-Adress-Befehle, da immer (höchstens) ein Operand adressiert werden muss. Es gibt jedoch auch Mehr-Adress-Befehle, so z.B. Zwei-Adress-Befehle, insbesondere, wenn mehr als ein Akku vorhanden ist, da dieser dann explizit angegeben werden muss.
- i) Da beim Inhalt einer Speicherzelle als Bitfolge nicht erkennbar ist, ob es sich hierbei um eine Adresse oder einen Datenwert handelt, muss der Rechner aufgrund des zeitlichen Kontextes selbst entscheiden, wie eine spezielle Bitfolge zu interpretieren ist. Dieses Problem wird technisch durch das Zwei-Phasen-Konzept gelöst.
- j) Der von Neumannsche Flaschenhals umschreibt das Problem, dass die Geschwindigkeit, mit welcher auf einen Speicher zugegriffen werden kann, mit der einer CPU nicht vergleichbar ist. Früher war die CPU zu langsam heute ist der Speicherzugriff zu langsam. Später wurde versucht die CPU durch komplexe Instruktionen stärker zu belasten als etwa einen Bus, um dies auszugleichen.

Problem 3: Ein MMIX-Programm (1+1+1+1+5=9)

Diese Aufgabe soll in die Benutzung der MMIX-Umgebung einführen.

- a) Erzeugen Sie die Datei `hello.mms`, die folgendes Programm enthält:

```
msg  BYTE  „Hello World“,#A,0      % Nachricht
Main  GETA  $255,msg                % Nachrichtenadresse nach $255
      TRAP  0,Fputs,StdOut          % Ausgabe aush $255
      TRAP  0,Halt,0                % Programm beenden
```

(Beachten Sie, dass Zeilen, die nicht mit einem Label beginnen, mit einem Leerzeichen oder einem Tab beginnen müssen.)

- b) Kompilieren Sie das Programm mit Hilfe eines Aufrufes von `mmixal.exe hello.mms`! Welche Ausgabedatei entsteht daraus?
- c) Führen Sie das Programm mit Hilfe eines Aufrufes von `mmix.exe hello aus`! Welche Ausgabe erscheint?
- d) Was ist die Auswirkung von „`#A`“ bei der Definition von `msg`?
- e) Fügen Sie die folgende Zeile an den Anfang des Programms ein:

```
times IS      3
```

Wozu führt diese Zeile?

Schreiben Sie das Programm so um, dass die Nachricht `msg` genau `times` mal ausgegeben wird (unabhängig vom konkreten Wert von `times`). Alle hierfür benötigten Assembleranweisungen finden Sie im MMIX-CrashCourse auf den Seiten 8-15, den Befehl `BZ` (branch if zero) z.B. auf Seite 15.

TI II

Sommersemester 2010
PD Dr. Katinka Wolter

Problem 4: Ein MMIX-Program (0)

Mache Dich mit der Benutzung von `mmix` vertraut. Ausreichende Dokumentation dazu ist im WWW vorhanden. Auch in `mmix.tar.gz` sind einführende Texte enthalten (die auch im WWW vorhanden sind).

1. Erzeuge die Datei `einfach.mms`, die folgendes Programm enthält (Zeilennummern und Kommentare nicht mit eingeben):

```
1      LOC #100           Start des Programms im Speicher
2 x    IS $42            Anweisung, 'x' im Programm durch $42 zu ersetzen
3 y    IS $43           ...
4 z    IS $44           ...
5 Main SET x,40         Main: hier beginnt das Hauptprogramm
                          Setze x (also das Register 42, $42) auf den Wert 40
6      SET y,7           Setze y auf 7
7      MUL $44,$42,$43  Multipliziere Reg. 42 mit Reg. 43,
                          schreibe das Ergebnis nach Reg. 44
8      ADD z,z,3        ...
9      DIV y,z,13       ...
10     TRAP 0,Halt,0    Ende des Programms
```

2. Erzeuge den assemblierten Code `einfach.mmo`.
3. Welchen Wert enthalten die Register 42, 43 und 44 nach der Ausführung von Zeile 9 durch MMIX? Wie kommt man an diese Information bei Benutzung von `mmix`? (Außer durch Ausrechnen im Kopf.)
4. Ändere das Programm so ab, dass
 - anfangs $x = 6$, $y = 7$ und $z = 20$ gilt,
 - der Term $xy(x+y)/(z-y)$ berechnet wird und das Ergebnis in Register 14 steht.

Problem 5: Einfache Ein-/Ausgabe (4+4+4+4=16)

- a) Das folgende Programm `sehr_einfache_Ausgabe.mms` gibt den Wert des Registers `x` als positive, ganze Zahl mit höchstens vier Stellen aus:

```
Buffer BYTE " 0",#a,0   vier Zeichen, newline, String-Terminator 0
x      IS $14
digit  IS $42
      LOC #100
Main   SET x,242
      GETA $17,Buffer   get address of Buffer -> $17
      SET $18,3         $18 zaehlt die Stellen herunter 3...0
Loop   DIV x,x,10       durch 10 teilen um die letzte Stelle
                          als Rest zu erhalten
      GET digit,rR     der Rest der Division steht in rR -> digit
      ADD digit,digit,'0' addiere den Wert des ASCII Zeichens '0'
      STBU digit,$17,$18 store byte unsigned, schreibe das Zeichen
                          in den Buffer mit dem Offset $18 (3...0)
      SUB $18,$18,1    Offset herunterzaehlen
      BNZ x,Loop      branch if x not zero -> loop
      SET $255,$17    Adresse des Buffer in $255 laden
      TRAP 0,Fputs,StdOut String, auf den $255 zeigt, ausgeben
      TRAP 0,Halt,0
```

TI II

Sommersemester 2010
PD Dr. Katinka Wolter

Was passiert bei Zahlen, die dezimal mehr als vier Stellen benötigen? Wie kann man verhindern, dass außerhalb des Buffers geschrieben wird? Ändern Sie das Programm entsprechend ab. (Dabei sollen von allen Zahlen mit mehr Stellen nur die letzten vier Stellen ausgegeben werden.)

- b) Ändern Sie das Programm so ab, dass die Zahlen im Dualsystem ausgegeben werden.
- c) Ändern Sie das Programm so ab, dass auch negative Zahlen ausgegeben werden können.
- d) Das folgende Programm `sehr_einfache_Eingabe.mms` benutzt zwei verschiedene Arten zur Eingabe von Daten. Wenn ein Kommandozeilenparameter übergeben wird (z.B. durch `mmix sehr_einfache_Eingabe Hallo`), so wird der Programmname (der immer der erste Parameter ist) und der erste weitere Parameter ausgegeben. Ansonsten wird von `StdIn` (üblicherweise die Tastatur) eine Zeichenfolge gelesen und die ersten 7 Zeichen werden ausgegeben.

Eingabe OCTA 0,8		Platz fuer die Uebergabe der Buffer-Adresse es sollen 8 Zeichen gelesen werden inkl. terminierender 0
Buffer BYTE " "		an diese Stelle wird die Eingabe geschrieben
newline BYTE #a,0		ein newline-Zeichen mit String-Terminator
BufAddr IS \$42		
EinAddr IS \$43		
Vergl IS \$41		
LOC #100		
Main CMP Vergl,\$0,1		compare, ist die Anzahl der Kommandozeilen- -Parameter groesser als 1 (steht in \$0)
BP Vergl,option2		Wenn ja (Ergebnis positiv) gehe zu option2
option1 GETA BufAddr,Buffer		sonst hier weiter, Adressen bestimmen
GETA EinAddr,Eingabe		
STOU BufAddr,EinAddr,0		Speichere die Adresse des Buffer bei 'Eingabe'
SET \$255,EinAddr		Setze Register \$255
TRAP 0,Fgets,StdIn		Lies String von StdIn nach Buffer
SET \$255,BufAddr		
TRAP 0,Fputs,StdOut		und gib ihn wieder aus
GETA \$255,newline		
TRAP 0,Fputs,StdOut		gefolgt von newline
TRAP 0,Halt,0		
option2 LDOU \$255,\$1,0		lies die Adresse des ersten Kommandozeilen- -Parameters in \$255 (\$1 zeigt darauf)
TRAP 0,Fputs,StdOut		und gib diesen aus
GETA \$255,newline		
TRAP 0,Fputs,StdOut		
LDOU \$255,\$1,8		die Adresse des zweiten Parameters steht 8 Byte weiter, lies sie ein ...
TRAP 0,Fputs,StdOut		
GETA \$255,newline		
TRAP 0,Fputs,StdOut		
TRAP 0,Halt,0		

Vollziehen Sie das Programm nach und schreiben Sie mit diesen Informationen ein Programm, das eine kleine, positive, ganze Zahl von `StdIn` einliest und die Zeichenkette Rechnerstrukturen ausgibt, wenn diese Zahl gleich 42 ist, sonst nichts tut.

TI II

Sommersemester 2010
PD Dr. Katinka Wolter

Problem 6: Rechnen in verschiedenen Zahlendarstellungen (8)

Berechnen Sie die Ergebnisse (Ergebnis zur Basis 10 angeben).

- a) $0x45 + 13444_8 - 10001110_2 + 11110_{10}$
- b) $111011_2 * 2^2 + 11110_2 * 2^4 - 101_2 * 2^2 * 11_2 * 2^1$
- c) $11010100000_2 * 2^{-3} + 332545_8 * 8_{10} - 0xFEEF$
- d) $1101.1001_2 + 11.11_2 + 111.11100_2 * 2^{-3}$

Achtung! Der Rechenweg muss bei allen Aufgaben stets erkennbar sein.

Solution 3

- a) TODO