TECHNISCHE UNIVERSITÄT DRESDEN

VLSI-EDA

Fakultät Informatik, Institut für Technische Informatik, Professur VLSI-Entwurfssysteme, Diagnostik und Architektur

Rechnerstrukturen und -organisation (RSO)

Assembler-sprache

Rainer G. Spallek

TU Dresden, 03.02.2021

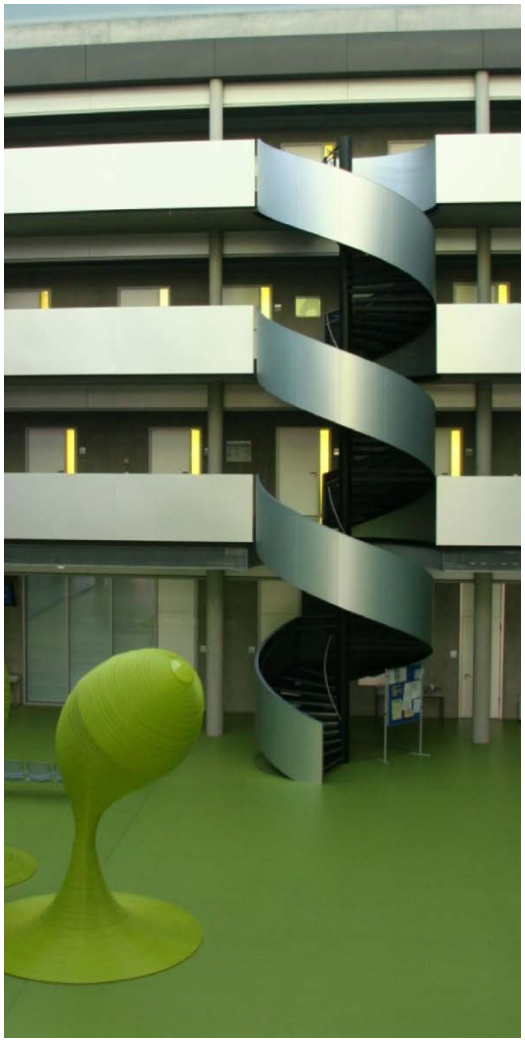
Rainer.Spallek@tu-dresden.de http://vlsi-eda.inf.tu-dresden.de

Gliederung

1 Zielstellung

2 Assemblersprache - Assembler

3 Zusammenfassung



1 Zielstellung

- Methodenverständnis zur Generierung eines Maschinenprogramms.
- Kennenlernen eine hardwarenahen Programmiersprache zur Erzeugung eines Maschinenprogramms.
- Assemblersprache und Assembler als low-level language zur maschinennahen Programmierung, Maschinenprogrammgenerierung.
- Was ist bei der Assemblerprogrammierung zu beachten.
- Programmierung an einem einfachen Beispielprogramm.

2 Assemblersprache - Assembler

- **Assembler:** Setzt die Assemblersprache (Programmiersprache) direkt in ein Maschinenprogramm (Maschinenbefehle) um.
  → Ein Assemblerbefehl entspricht dabei einem Maschinenbefehl.
- **Assemblersprache:** (low-level languages), hardwarenahe Programmierung, teilweise effizienter und detaillierter → aber unflexibler.
- HLL (high-level languages), hardwareunabhängige Programmierung. Compiler übernehmen die Erzeugung der Maschinenprogramme. Möglichkeit der Einbettung von Assemblersprache → Inline-Assembler.
- Verschiedene Prozessorarchitekturen haben auch verschiedene Assembler- und Maschinensprachen, kein Standard.
- **Makroassembler:** Ermöglicht parametrisierbare Assemblerprogrammierung.
  → Eine Makroanweisung kann mehrere Maschinenbefehle generieren.
- **Cross-Assembler:** Generiert auf einem Host Code für andere Zielplattform.
- **Disassembler:** Setzten Maschinensprache wieder in Assemblersprache um.

Aufgaben eines Assemblers

- Umsetzen von Befehlsmnemonics in der Anweisung einer Assemblersprache, z.B. in binären opcode.
- Umsetzen von Datenmnemonics in deren binäre Repräsentation.
- Verwaltung von Konstanten und von Adressen von Befehlen, Daten und Sprungzielen.
- Verwaltung von Verzweigungen und Berechnung der Sprungziele.
- Erzeugen von Maschinencode, ggf. als Objektdateien für zweistufige Übersetzungsprozesse mit Linker und der Möglichkeit zur Einbindung weiterer Programmteile (z. B. Unterprogramme) aus Bibliotheken.
- Ermöglichen von Debugging und Test.
- Generieren eines Assembler-Listings, Fehlernachrichten, etc.

→ Variablenvereinbarungen, Datenstrukturen, Datentypen, Byte-Reihenfolge (byte order), Speicherausrichtung (data alignment), Pipeline-Konflikte, Unterbrechungen (exeptions) … müssen vom Programmierer in der Assemblersprache explizit berücksichtigt werden.

Assemblersprache

Jede Architektur und jeder Assembler spricht eine eigene Sprache!

Assembler-Befehlszeile (http://www.inf.fu-berlin.de/lehre/SS00/19502-V/spimdoku.pdf)

Marke: Befehl Arg1, Arg2, Arg3 # Kommentar

Marke:	Marke wird mit : abgeschlossen, z.B. bei Verzweigungen die Zielmarke.
Befehl:	Der eigentliche Assemblerbefehl aus der Assemblersprache.
Arg:	Die Argumente des Befehls, 0 bis 3 Argumente Arg1, Arg2, Arg3 üblich, durch Komma oder Leerzeichen getrennt.
Kommentar:	Der Kommentar wird mit vorgestelltem # gekennzeichnet.
Pseudobefehle:	Entsprechen keinem direkten Assemblerbefehl, sondern generieren vielmehr eine Folge von Befehlen, z.B. Makro-Assembler.

Struktur eines Assembler-Programmes

	Marke	Befehl	Argumente			Kommentare	Mnemonik
.data							**Datensegment**
	x:	.word	10			# x:= 10	Datentyp und
	y:	.word	95			# y:= 20	Belegung
	z:	.word	0			# z:= 0	
.text							**Textsegment**
	main:	lw	R1,	x		# R1:= x	Hauptprogramm
		lw	R2,	y		# R2:= y	Lade Wort
	M1:	add	R3,	R3,	R1	# R3:= R3+R1	Addiere
		bge	R3,	R2,	M2	# →M2 bei R3≥R2	Sprung bei ≥
		j	M1			# → M1 (jump)	unbed. Sprung
	M2:	sw	R3,	z		# z:= R3	Speichere Wort
		syscall				# → OS call	Betriebssystemruf

Assembler Beispielaufgabe

Gegeben sei folgende Befehlscodierung:

Befehl	Spezifikation	Bemerkung
MOV Ri, Rj	Ri ← Rj	Registertransfer
ADD Ri, Rj	Ri ← Ri + Rj	Addition, Integer
CLR Ri	Ri ← 0	Nullung, Löschen
NOT Ri	Ri ← /Ri	bittweise Negation
INC Ri	Ri ← Ri + 1	1-Inkrement

Der Prozessor hat nur die Register R0, R1, R2, R3 und verfügt nicht über Pipelining. Implementieren Sie mit Hilfe dieses Befehlsumfanges die Berechnung des folgenden Ausdrucks:

$$y = -4a + 2b - 2$$

Die Operanden a und b stehen in dieser Reihenfolge in den Registern R1 und R2 zur Verfügung. Das Ergebnis soll zum Abschluss der Rechnung wieder im Register R1 stehen. Alle Variablen und Operationen sind vom Typ INTEGER.

Mögliche Lösungvarianten

Befehl	Kommentar
NOT R1	R1:= -a-1
INC R1	R1:= -a
ADD R1, R1	R1:= -2a
ADD R1, R1	R1:= -4a
ADD R2, R2	R2:= 2b
ADD R1, R2	R1:= -4a+2b
CLR R2	R2:= 0
NOT R2	R2:= -1
ADD R2, R2	R2:= -2
ADD R1, R2	R1:= -4a+2b-2

Befehl	Kommentar
ADD R1, R1	R1:= 2a
NOT R1	R1:= -2a-1
ADD R1, R2	R1:= -2a+b-1
ADD R1, R1	R1:= -4a+2b-2

3 Zusammenfassung

- Assemblersprache und Assembler sind nicht einheitlich, nicht standardisiert.
- Ein Assembler generiert aus einer Befehlsfolge in Assemblersprache ein Maschinenprogramm, Maschinenbefehle.
- Für die Assemblerprogrammierung sind detaillierte Kenntnisse der Ziel-Hardware erforderlich, Adressierungstechniken, Pipelinekonflikte, Parallelität, …
- Ein Assemblerbefehl generiert genau einen Maschinenbefehl.
- Sehr aufwändige Programmierungstechnik, hoher Zeit- und Ressourcenaufwand (Spezialisten).
- Mit Assemblerprogrammierung lässt sich der Befehlsvorrat eines Prozessors voll ausnutzen.