# Programmierung Mikrocomputersysteme

## Ausgangslage

Maschinennahe Programmierung eignet sich im Bereich von Steuerungen und Regelungen besonders gut. Einerseits können die Programme sehr schlank gehalten werden und andererseits hat man als Programmierer hat man so direkt Einfluss auf die Hardware, ohne dass auf irgendein zwischengeschaltetes Betriebssystem Rücksicht genommen werden muss.

Um eine kleine Idee von dieser rationellen und ressourcenschonenden Art der Programmierung zu bekommen, wollen wir mit einer Mikroprozessor-Simulation ein paar einfache Steuerungsprogramme mit Assembler realisieren.

Eine Assemblersprache ist eine spezielle Programmiersprache, welche die Maschinensprache einer spezifischen Prozessorarchitektur in einer für den Menschen lesbaren Form repräsentiert. Jede Computerarchitektur hat folglich ihre eigene Assemblersprache.

Der Mikroprozessor-Simulator besteht aus folgenden Elementen:

|  |  |
| --- | --- |
| * CPU * 256 Bytes RAM * 16 I/O-Ports * Hardware-Timer * Keyboard |  |
| * verschiedene (simulierte) Peripheriegeräte auf den Ports 0 bis 6 |  |
| * Assembler-Umgebung |  |

vereinfachte Architektur des Mikroprozessor-Simulators basierend auf dem Intel 9086 Chips

## Auftrag 1 - Grundlagen

Kopieren Sie die Dateien für den Mikroprozessor-Simulator aus dem Modulordner[[1]](#footnote-1) in das Verzeichnis C:\m121\cpu-simulator der virtuellen Maschine *bmWP5*. Erstellen Sie vom Programm C:\m121\cpu-simulator\sms32v50.exe eine Verknüpfung auf den Desktop.

Starten Sie das Programm und laden Sie das erste Beispielprogramm 01FIRST.ASM aus dem Verzeichnis C:\m121\cpu-simulator\asm\. Lassen Sie das Programm schrittweise ablaufen. Beantworten Sie mit der Online-Hilfe folgende Fragen:

|  |  |  |
| --- | --- | --- |
| Was steht nach Ausführung des Programms in den verschiedenen Registern?  **Online-Hilfe** F1 -> 01 First Program |  | |
| Bedeutung der Register AL, BL, CL, DL?  **Online-Hilfe** F1 -> Instruction Set Summary | Universelle acht Bit Register. Die Register können vorzeichenlose Zahlen von 0 bis +255 oder Zahlen mit Vorzeichen von - 128 bis +127 aufnehmen. | |
| Was ist der Maschinencode der nebenstehenden Assembleranweisungen?  **Online-Hilfe** F1 -> Instruction Set Summary | MOV AL,04  MOV BL,02  ADD AL,BL | D0 00 04  D1 01 02  A0 00 01 |
| Wie viele *Fetch‘s* werden benötigt um den Befehl MOV AL zu laden? | 2 Fetch | |
| Um wieviele Schritte verschiebt sich der Programmzähler IP nach MOV AL,04? | Der Programmzähler IP wird um 3 hochgezählt | |

Ändern Sie das Programm 01FIRST.ASM wie folgt:

; ===== WORK OUT 2 PLUS 2 ======================================

CLO ; Close unwanted windows.

MOV AL,2 ; Copy a 2 into the AL register.

MOV BL,**7F** ; Copy a **127** into the BL register.

ADD AL,BL ; Add AL to BL. Answer goes into AL.

END ; Program ends

; ===== Program Ends ===========================================

|  |  |
| --- | --- |
| Wie ist der Inhalt der Register AL und SR nach Ausführung des Programms?  **Online-Hilfe** F1 -> Negativ Numbers F1 -> Architecture |  |
| Was ist die Bedeutung der Flags I, S, O, Z im SR-Register?  **Online-Hilfe** F1 -> Glossary | Zeroflag Z wird bei Nullresultaten (z.B. 2 x 0) gesetzt  Overflow-Flag O wird bei Registerüberlauf gesetzt.  Sign-Flag S wird bei negativen Resultaten gesetzt.  Interrupt-Flag I wird zur Abhandlung von Programm-Interrupts benutzt. |

## Auftrag 2 – Ampelsteuerung

Mit der folgenden Übung soll die die Ampel gemäss Schweizer Strassengesetz gesteuert werden.

|  |  |
| --- | --- |
| Die Ampel wird über den I/O-Port 01 mit den Bit 7 – 2 gesteuert. |  |
| Ergänzen Sie die nebenstehende Tabelle mit den Bit-Kombinationen:  rot rot  rot rot/orange  rot grün  rot grün/orange  rot rot  rot/orange rot  grün rot  grün/orange rot |  |

Schreiben Sie mit Hilfe des untenstehenden Codes ein Programm, welches die zwei Ampeln nach folgenden Regeln steuert:

* rot 🡺 rot+gelb 🡺 grün / grün 🡺 gelb 🡺 rot

; ===== Ampelsteuerung ======================================

CLO ; unerwünschte Fenster schliessen

Start:

MOV AL,90 ; Kopiere 1001 0000 ins AL-Register

OUT 01 ; schicke den Inhalt von AL auf Port 01 (Ampel)

MOV AL,98 ; Kopiere 1001 1000 ins AL-Register

OUT 01 ; schicke den Inhalt von AL auf Port 01 (Ampel)

JMP Start ; springe zurück zum Start

END

; ===== Programm Ende =========================

|  |  |
| --- | --- |
| Vereinfachen Sie das Programm, indem Sie für die Ampelphasen Datentabellen einführen.  **Online-Hilfe** F1 -> 08 Data Tables | ; Ampelsteuerung  JMP Start  ; Datenbereich  DB 90  DB 98  Start:  MOV BL,2 ;Adresse 02 Start Datenbereich  MOV AL,[BL] ;Kopier Datenbereich nach AL  OUT 01 ;Wert von AL an Port 01 |
| Für die ganz Schnellen: Erweitern Sie das Programm mit verschiedenen Wartephasen!  **Online-Hilfe** F1 -> 06 Procedure | rechts grün – links rot → lang  rechts rot – links grün → mittellang |

## Auftrag 3 – Schrittmotor

Mit der nächsten Übung soll ein Schrittmotor mit Ganz- und Halbschritten betrachtet und gesteuert werden.

|  |  |
| --- | --- |
| Schematische Darstellung eines Schritt-Motors. Die animierten Grafiken für einen Ganzschrittmotor sowie für einen Halbschrittmotor finden Sie im Verzeichnis 99\_Diverses des Modulordners   * ani121\_ganzschritt.gif * ani121\_halbschritt.gif   Studieren Sie die Bewegungsabläufe der zwei Motoren. |  |
| Der Motor in der Simulation wird über den I/O-Port 05 mit den Bit 3 – 0 gesteuert. |  |
| Ergänzen Sie die Tabelle für eine Vorwärtsbewegung in Ganzschritten.  Die Ausgangslage ist das Bitmuster  0000 0001 |  |
| Ergänzen Sie die Tabelle für eine Vorwärtsbewegung in Halbschritten.  Die Ausgangslage ist das Bitmuster  0000 0001 |  |
| Programmieren Sie den Schrittmotor sowohl für Ganz- wie auch für Halbschritte. Stellen Sie die nötigen Werte wieder in einen Datenbereich.  **Hinweis** Benutzen Sie das Programm aus der Übung 1 als Ausgangslage. | * nur Ganzschritte * nur Halbschritte * abwechslungsweise Ganz- und Halbschritte |

## Auftrag 3 – Seven Segement

Mit der folgenden Übung schauen wir die Steuerung einer doppelten Seven Segment-Anzeige an.

|  |  |
| --- | --- |
| Die Seven Segment-Anzeige wird über den I/O-Port 02 mit den Bit 7 – 1 gesteuert. Mit dem Bit 0 wird entschieden, ob die linke oder die rechte Anzeige bedient werden soll.   * Bit 0 Wert 0 → linke Ziffer * Bit 0 Wert 1 → rechte Ziffer |  |
| Ergänzen Sie die nebenstehende Tabelle mit den korrekten Binär- und Hexwerten für die linke Ziffer.  Für die Ansteuerung der rechten Ziffer muss jeweils nur der Wert 1 addiert werden. |  |
| Programmieren Sie einen Count-Down von 09 – 00. Bei 00 angelangt, sollen die Ziffern 00 mehrmals blinken. Stellen Sie die nötigen Werte in einen Datenbereich. | d:\temp\apptemp\SNAGHTML12dafe3.PNGd:\temp\apptemp\SNAGHTML12e3e3d.PNGd:\temp\apptemp\SNAGHTML12e83d5.PNGd:\temp\apptemp\SNAGHTML12f08ad.PNGd:\temp\apptemp\SNAGHTML12e83d5.PNGd:\temp\apptemp\SNAGHTML12f08ad.PNG |

1. http://www2.gibb.ch/iet/module/dokumente/modul121/98\_Software/cpu-simulator [↑](#footnote-ref-1)