Programmieren in Assembler

Marcel Waldvogel

http://www.erg.abdn.ac.uk/users/gorry/eg2
http://whatis.techtarget.com/definition/0,,si

Hierarchie

- Konzeptionelle Ebene
  - Implementation
- Hochsprachenebene
  - Compiler
- Assemblersprachenebene
  - Assembler
- Betriebssystemebene
  - Partielle Interpretation
- Maschinenebene
  - Interpretation
- Mikroprogrammebene
  - Direkte Ausführung
- Hardwareebene

Assembler vs. Assembler

- C vs. C? Pascal vs. Pascal? Nein.
- Konzeptionelle Ideen übertragbar
- Assemblerprogramm lesen oder schreiben: Handbuch kaufen/herunterladen
  - Prozessor und Hilfschips
- Einsatz von Assembler
  - Betriebssysteme
  - Leistungssteigerung
  - Compilerbau (inklusive JIT etc.)
  - Analysieren von Programmen ohne Quelltext

Beispielprozessor: MIPS

- "Typischer" RISC-Vertreter
  - Kleinster gemeinsamer Nenner
  - Keine Details, leichte Vereinfachungen
- 32 Integerregister (r0 . . . r31)
  - r0=0
- 32 Fliesskommaregister
- Flags
  - Zero
  - Negative
  - Carry
  - oVerflow
- Stapelspeicher

Befehlssatz

- Standardbefehle (ADD, SUB, ...): Register
  - ADD r3, r1, r2 # r3 := r1+r2
  - SUB r3, r1, r2 # r3 := r1-r2
- Standardbefehle: Register-Immediate
  - ADDI r3, r1, #5 # r3 := r1+5
  - Wertebereich: 16 Bit
- Lade-/Speicherbefehle
  - LD r3, (r1) # r3 := Mem[r1]
  - ST r3, 8(r1) # Mem[r1+8] := r3
- Laden von Konstanten
  - LDA 5000(r2), r3 # r3 := 5000+r2
  - LDA 5000(r0), r3 # r3 := 5000

Test und Sprung

- Vergleiche
  - SUB r0, r1, r2 # Vergleiche r1 und r2
  - ADD r0, r1, r0 # Vergleiche r1 mit 0
- Bedingte Sprünge
  - BEQ marke # Springe zu "marke", wenn =
  - BNE, BGT, BLT, BGE, BLE, ...
- Unbedingte Sprünge
  - J ziel
  - JAL unterprogramm
  - RET # Rückkehr Unterprogramm

Beispiel: Bits Zählen

- Aufgabenstellung
  - Wieviele Bits eines Registers haben Wert 1?
  - Eingabe: r1, Ausgabe: r2
- Ansatz
  1. Höherer Programmiersprache, keine Tricks
  2. Umsetzen nach Assembler
  3. Testen
  4. Optimieren

Von C nach Assembler

```
int bitcount(int bits) {
   int i;
   int count = 0;
   for (i = 0;
            i < 32;
        i++) {

        if (bits[i] == 1) {
           count += 1;
      }

   }
   return count;
}
```

```
int bitcount(int bits) {
   int i = 0;
   int count = 0;
   do {


      int bit_i = (bits & (1 << i);
      if (bit_i != 0) {
            count += 1;
      }
      i += 1;
   } while (i < 32);
   return count;
}
```

Von C nach Assembler (2)

```
int bitcount(int bits) {
   int i = 0;
   int count = 0;
   do {

      int bit_i = (bits & (1 << i);
      if (bit_i != 0) {
            count += 1;
      }
      i += 1;
   } while (i < 32);

   return count;
}
```

```
bitcount:
      ADD   r2, r0, r0
      ADD   r3, r0, r0
loop:
      ADDI  r4, r0, #1
      SHLL  r4, r4, r2
      AND   r4, r4, r1
      BEQZ  noadd
      ADDI  r2, r0, #1
noadd:
      ADDI  r2, r0, #1
      SUBI  r0, r2, #32
      BNEZ  loop
      RET
```

MIPS vs. Intel

```
bitcount:
        ADD     r2, r0, r0
        ADD     r3, r0, r0
loop:
        ADDI    r4, r0, #1
        SHLL    r4, r4, r2
        AND     r4, r4, r1
        BEQZ    noadd
        ADDI    r2, r0, #1
noadd:
        ADDI    r2, r0, #1
        SUBI    r0, r2, #32
        BNEZ    loop
        RET
```

```
bitcount: # EAX->EBX
        XOR     EBX, EBX
        XOR     CL, CL
loop:
        MOV     EDX, #1
        SHL     EDX, CL
        AND     EDX, EAX
        BEQ     noadd
        INC     EBX
noadd:
        INC     CL
        CMP     CL, #32
        BNE     loop
        RET
```

Weitere Beispiele

- Addition zweier Variablen
  - Adressen konstant
  - Adressen in Registern
    - Anwendung: Multiplikation
- Vektoraddition
  - Rückwärts
- 3n+1
  - Zählen
  - Unterprogramm

Programmiertipps

- Nachschlagetabellen
- Dekrementieren bei Schleifenzählern
  - Spart Test und oft ein Register
- Bitfieseleien oft schneller als Sprünge
  - Sprünge teuer (oft 10-20 Instruktionen)
  - Conditional Moves (alle modernen CPUs) bzw. Conditional Opcodes (ARM)
    - Bits zählen: ADD with Carry
    - Wert mit sich selbst überschreiben (fast) gratis
    - Befehl ausführen und Nebeneffekt kompensieren: $3n+1 = 6(n >> 1)+4$, wenn n ungerade
    - Datenstrukturen linearisieren: Matrixaddition
- Werte frühzeitig bestimmen
  - Instruktionsumordnung des Prozessors

Architekturtipps

- Registerfenster (SPARC)
  - Je 8 Register Input, Local, Output und Global
- Mehrere Bedingungsvariablen (ARM)
  - Flags in jedem Register möglich (Alpha)
- Verzögerungen
  - Branch Delay Slot (SPARC)
  - Load/Store Delay (MIPS)

Besonderheiten

- Usermodus
  - Segmente und -präfixe (x86)
  - Wiederhol-/Stringbefehle (x86, m68k)
  - Prefetch, Cache Management
- Systemmodus
  - Unzählige spezifische Kontrollregister
    - (Data) Breakpoint, Zähler
  - Memory Management/Virtual Memory
    - Zugriff auf physikalische Adressen
    - Adressen aus Sicht anderer Prozessoren/Karten

Abkürzungen

- B, BR = Branch; J, JMP = Jump
  - Short und Long; (un)bedingt
- B = Byte, danach z.B.
  - W = Word (16 Bits), L = Long (32 Bits), Q = Quadword (64 Bits)
  - H = Halfword (16), W = Word (32), D = Doubleword (64)
- Segmentadressierung: Near, Far (x86)