1 Unterprogrammaufruf

Fragen:
- wo und wie liegen die Parameter beim Aufruf (→ Calling Convention)
- wo liegt der Rueckgabewert nach Ruecksprung (→ Calling Convention)

Optionen:
- Register oder Stack (Reihenfolge wichtig)
- bei Stacknutzung: wer entfernt die Parameter

1.1 Calling Convention

Die Calling Convention ist ein Vertrag zwischen allen Entwicklern auf einem System. Sie bestimmt wie Parameter und lokale Variablen uebergeben werden und der Stack/Register organisiert sind.
Sie wird nicht von der Hardware kontrolliert. Jedoch koennen bestimmte Befehle auf die jeweilige Hardware optimiert sein.

→ abhaengig von ISA und Betriebssystem

1.2 IA-32 Konvention

- Parameter, lokale Variablen auf dem Stack gesichert durch den Caller via PUSH
- Call legt BZ auf den Stack → nach Unterprogrammaufruf ebp Basispointer zur Adressierung von Variablen
- genaue lokale Platzeinteilung haengt vom Compiler ab
- Rueckgabewert in eax falls Integer
- Aufrufer entfernt Parameter nach Ruecksprung

→ innerhalb von Unterprogrammen mehr Flexibilitaet (keine Aussenwirkung)
→ esp muss immer richtig stehen (Schutz des Stacks)

1.3 Registersicherung

Register die vom Unterprogramm gesichert werden:

- "Callee-saved"
- nach Aufruf unveraendert
- ebx, esi, edi, ebp, esp

Register, die vom Aufrufer gesichert werden

- "Caller-saved"
- koennen von Unterprogrammen veraendert werden
- eax, edx, ecx, condition flags

2 IA-32 Ergaenzungen

- VMX: Virtual Machine Support
- F16C: half precision floating-point conversion
- Transaktions-Memory: Spekulation fuer atomare Zugriffe
- SIMD Instruktionen: Single Instruction Multiple Data → Parallelitaet
- SSE bringt 128bit Register
- AVX bringt 256bit Register

3 Optimierungen

- O flags fuer Optimierung (O2 safe, O3 unsafe)
- O3 kann die Semantik des Programmcodes umschreiben um effektiver und schneller zu sein
- Optimierungen koennen Schleifen zur Compilezeit vorrechnen, wenn alle benoetigten Rahmenfaktoren zur Compilezeit bereits bekannt sind fuer den Compiler

4 ARM Familie

ARM stellt Designs fuer eine Reihe von ARM Kernen/Prozessoren her:

- lizensiert an Partner
- keine eigene Herstellung
- arbeiten an der ARM Umgebung

→ somit unterschiedliche Hersteller aber selbe Kernarchitektur
→ Ergebnis: breite Reihe an ARM Prozessoren

- ARMv8 (AArch64) - erste ARM 64bit ISA → z.B. Raspberry 3

ARMv8 ISA:

- RISC (Format: 4 Bytes pro Maschinenbefehl)
- Load/Store-Architektur (Register-Register-Maschine)
- arithmetische/logische Operationen kombiniert mit "Barrel-Shifter"

ARMv8 ISA Besonderheiten:

- Rueckwaertskompatibel bis zu ARMv5 (im 32bit Modus)
- 32bit Groesse fuer Maschinenbefehle - Konsequenz:
keine beliebige Konstantenkodierung moeglich
- relative Sprungadressen +/- 24bit vom BZ (+/- 32MB)
- absolute Sprungadressen nur via Register
- laengere Konstante aus Speicher via Konstantenpool
- alternative Konstruktion mit Shift

4.1 ARMv8 Formate und Register

Unterstuetzung von Integer mit/ohne Vorzeichen und IEEE 754 Fliesskomma:

- Integer bis 64bit
- Floating-point: 32/64bit (spaeter auch FP16)

Register

- 31 64bit Register zur allgemeinen Verwaendung (mehr als x86)
- 31 128bit SIMD Register
- 64-bit PC und SP
- "Zero" Register

Speichersystem:

- virtuelle Adresse 64bit
- unterstuetzt "Little Endian" und "Big Endian"
- "unaligned" Zugriffe eingeschraenkt
- Stackpointer muss 16Byte aligned sein → sonst Exception
- Seitenbasierter virtueller Speicher (Seitengroesse von 4KiB bis 1GiB)

4.2 ARMv8 Calling Convention