

Debugging & Co.

objdump

objdump Programm, um verschiedene Informationen über Objekt-Dateien anzuzeigen (auch Assemblercode).

-D <datei>

-M <Syntax-Typ> Intel oder AT&T Syntax verwenden

GDB

gdb -q <Pfad zu Datei> Führt GNU-Debugger aus

set args <arguments> Übergibt Argumente an das auszuführende Programm.

list Quellcode anzeigen.

disassemble <funktion> Disassemblierung einer Funktion.

GDB ausführen

run Programm in GDB starten.

kill Killt das laufende Programm.

quit GDB beenden.

Breakpoints

break <zeile> Stoppt bei Erreichen der genannten Zeile.

break <funktion> Stoppt beim Erreichen der Funktion. (nur C)

break <datei:zeile> Stoppt bei Zeile in spezifischer Datei.

delete <breakpoint #> Entfernt einen Breakpoint.

clear Entfernt alle Breakpoints.

enable <breakpoint #> Aktiviert deaktivierten Breakpoint.

disable <breakpoint #> Deaktiviert aktivierten Breakpoint.

Watchpoints

watch <zeile> Setze neuen Watchpoint bei Erreichen der genannten Zeile.

watch <funktion> Setze neuen Watchpoint bei Erreichen der genannten Funktion.

Debugging & Co. (cont)

watch <datei:zeile> Setze neuen Watchpoint bei Erreichen der genannten Zeile in spezifischer Datei.

Stepping

continue Weiterlaufen lassen bis zum nächsten Unterbruch.

next Nur eine einzelne Zeile ausführen, in der aktuellen Ebene bleiben.

step Führt die nächste Linie aus, springt aber in aufgerufene Funktionen.

finish Beendet die Funktion und kehrt zum Aufrufer zurück.

Examine

Untersuchung des Speichers

x/<n><f><u> x: examine
addr n: Wiederholungen, f: Anzeigeformat,
u: Einheitgröße (nfu optional)
addr: Adresse

<f>
o Oktale Darstellung
x Hexa. Darstellung
u Vorzeichenlose Darstellung
t Binäre Darstellung

<u>
b Einzelnes Byte
h Halbwort (2 Byte)
w Wort (4 Byte)
g Giant (8 Byte)

Informationen anzeigen

info registers Zeige Register bei aktuellem Breakpoint.

info register Zeige Info zu spez. Register.

<register>
kurz: i r
<register>

info args Zeige Argumente der Funktion bei aktuellem Breakpoint.



Debugging & Co. (cont)

`info breakpoints` Zeige Infos über Break- und Watchpoints.

`info threads` Zeige alle Threads an.

Einstellungen in GDB

`set dis intel` Disassembler-Syntax auf *intel* ändern.

`echo "set dis intel" > ~/.gdbinit` Speichere obige Einstellung dauerhaft.

Assembler

Allgemeine Register

Register sind für den Prozessor wie interne Variablen.

Der Prozessor unterscheidet zwischen verschiedenen Arten von Registern.

`eax / rax` Akkumulator-Register
Allgemein verwendbar, spezielle Bedeutung bei Arithmetikfehlern.

`ecx / rcx` Zähler-Register
Allgemein verwendbar, spezielle Bedeutung bei Schleifen.

`edx / rdx` Daten-Register
Allgemein verwendbar.

`ebx / rbx` Basis-Register
Allgemein verwendbar.

`esp / rsp` Stack Pointer

`ebp / rbp` Base Pointer

`esi / rsi` Source Index
Quelle für Stringoperationen.

`edi / rdi` Destination Index
Ziel für Stringoperationen.

Segmentregister

`cs` Codesegment

`ds` Datensegment

`ss` Stacksegment

`es` beliebiges Segment

`fs` beliebiges Segment

`gs` beliebiges Segment

Sonstige Register

`eip / rip` Instruction Pointer
zeigt auf aktuelle Instruktion, die der Prozessor gerade verarbeitet.

Assembler (cont)

`ef / RFLAGS` EFLAGS
Wird für Vergleiche und Speichersegmentierung verwendet.

Assembler Sprache

in Intel-Syntax

`operation` Format der Intel-Syntax

`<ziel>`,

`<quelle>`

Data Movement Instruktionen

`mov` bewegt Wert von Quelle ans Ziel.

`push` setzt seinen Operanden oben auf den hardwaregestützten Stack im Speicher.

`pop` entfernt das 4-Byte-Datenelement vom oberen Rand des hardwaregestützten Stacks in den angegebenen Operanden.

`lea` setzt die durch seinen zweiten Operanden angegebene Adresse in das Register des ersten Operanden.

Arithmetische und Logische Instruktionen

`add` addiert seine beiden Operanden zusammen und speichert das Ergebnis in seinem ersten Operanden.

`sub` siehe `add`, nur mit Subtraktion.

`inc` inkrementiert den Inhalt seines Operanden um eins.

`dec` dekrementiert den Inhalt seines Operanden um eins.

`imul` Integer Multiplikation

`idiv` Integer Division

`and`, `or`, `xor` Bitweise logisches UND, ODER & EXKLUSIVES ODER

`not` Bitweise logisches NOT

`neg` Negieren

`shl`, `shr` Bitshift nach links, Bitshift nach rechts.

Kontrollfluss-Instruktionen

`jmp` Sprung zu anderem Teil des Codes.

`j[kondition]` Konditioneller Sprung
Konditionen: `je`, `jne`, `jz`, `jc`, `jge`, `jl`, `jle`

`cmp` Vergleich von Werten

`call`, `ret` Diese Anweisungen implementieren einen Unterprogrammaufruf und -rückgabe.