Rechnernetze und -organisation

Lukas Prokop

11.10.26

Contents

1 Terminology

word fixed size sequence of bits defining a unit. Basically word defines the amount of bits to be processed at one time in an operation.

compilation translation of a program from source language to target language (eg. C to machine code)

32bit system Machine runs with memory addresses consisting of 32 bits

x86 Well-known family of Intel instruction set architectures which is deprecated nowadays in favor of amd64 / x86_64.

stack data structure implementing the LIFO (last in, first out) principle

2 GNU Compiler Collection

2.1 Compilation order

1. Write
2. Compile
3. Assemble
4. Link
5. Load
6. Execute

2.2 Procedure results

1. source code → C
2. preprocessing → C
3. compilated → ASM

4. assembled → object file
5. linked → executable

2.3 GCC options

-E create preprocessed code (step 1 and 2)

-S create assembler code.s (step 1, 2 and 3)

-c create object file.o (step 1, 2, 3 and 4)

-o specify output filename explicitly (not a.out)

-I Add the directory dir to the list of directories to be searched for header files.

-Wall Turn on all optional warnings

-x specify programming language explicitly

-std=gnu89 specify C standard to use

-O[**1230s**] Optimization modes

-static Static linking

-dynamic Dynamic linking

-m32 use 32bit architecture instead of current one

-mbig-endian Generate code for a big endian target (IA-64 only)

See also ld, cpp, objdump, gdb, as, ar, ranlib and readelf.

3 x86

3.1 Stack subroutine call

Stack runs against address 0 (in following figure: top is 0).

(ESP-1→) local var 2
local var 1
(EBP→) old EBP
old EIP
arg1
arg2
arg3

Table 1: Stack dump before subroutine call

3.1.1 Call (`call`)

1. store current EIP/PC on stack (ESP)
2. store old EBP on stack
3. allocate bytes for local variables
4. Let EIP/PC be pointer to subroutine
5. EBP is pointing to old EBP

3.1.2 Return (`ret`)

1. EBP = old EBP
2. EIP/PC = return address (old EIP/PC)
3. ”remove” function arguments from stack
   ESP = ESP + args

3.1.3 In words

All storage will be done on a new ”partition (frame)” on top of the stack. Therefore save the old EBP on top of stack. Let ESP be the new EBP (EBP = ESP). Subtract as many memory address from ESP to allocate memory for local variables. ESP has to keep the **top** of the stack.

3.1.4 Callee vs Caller cleanup

Callee cleanup

- + probably more space efficient
- - no variadic functions

Caller cleanup

- + variadic functions
- + default for x86 C compilers

3.2 x86 registers

3.2.1 List

8 general purpose registers:

EAX arithmetic results data

EBX (mov) pointer to data in data segment

ECX counter register for string operations

EDX port address for I/O – extending EAX for I/O

ESI source index

EDI destination index

EBP base pointer

ESP stack pointer

2 special registers:

EIP instruction pointer, program counter (PC)

EFLAGS results from comparisons/tests (implicit usage)

3.2.2 Description

ESP (Stack pointer) push & pop
points to top element of stack

EBP (Base/Frame pointer) references memory of current frame you are in
has to be manipulated explicitly

EIP (Instruction pointer) holds address of next instruction
Manipulated by jumps and call instructions

3.3 x86 addressing modes

indexed In "x(e)" you take the register behind x and jump as much bytes forward as register e is telling you

based Like indexed, but x is a constant

immediate Use parameter value of instruction directly / immediately

In direct mode a register is directly addressed (with its name). In indirect mode the content of a register is read (by surrounding parentheses).

3.4 ASM syntax

x86 is written in Intel syntax (Windows platform) or AT&T syntax (UNIX/Linux). We use the second. The following is a short cheatsheet:

op C(%A, %B, D) op *(%A + C + (%B * D))

op C(%A, %B) op *(%A + C + %B)

op C(%A), %B op %A + C, %B

op C(,%A,D), %B op C + %A * D, %B

3.5 Suffixes

q quad-word (64bit)

l long (32bit)

w word (16bit)

b byte (8bit)

3.6 Assembly directives

.string Allocate space for a global null-terminated string

.space Allocate bytes for a global

.rept / endr Make allocations between .rept and .endr multiple times (according to the parameter)

.long Allocate space for a global long

.equ Define equivalence (two names referring to the same)

.data Data section following. Defines global allocations.

.file Define metadata filename.

_start Label (for UNIX linker) of program entry point

.globl declare label as global

.text Text section following. Contains program logic.

.byte Allocate a single byte as a global

3.7 GNU assembler

-a Write to stdout

--gstabs include debugging information to output file. Can eg. be used as additional information for gdb.

Undefined symbols include references to other routines / functions which will be bound later on by the linker. Defined symbols are labels and routines already known to the compiler.

4 Sample boot program

1. check for signature
2. exit if signature not found
3. read start address
4. create temporary copy of start address
5. read amount of words to load
6. loop: read all words and store them to RAM
7. jump to start address
8. execution loop

5 Computation theory

5.1 Mealy automaton

current_state = f(prev_state)
output = out(input, current_state)

5.2 Moore automaton

current_state = f(prev_state)
output = out(current_state)

6 Abstraction

1. User
2. Application
3. Operating system
4. Architectural
5. Register transfer
6. Logical
7. Electrical
8. Physical

7 Pipelining

7.1 Dependencies

Data dependency If the first instruction is storing a value to a register which is used in the second instruction, the result is not going to be ready on time.

Control dependency If the first instruction includes a conditional jump, the following instruction might not be the one to be executed.

7.2 Possible solutions to pipelining chaos

- Compiler adds "No operation" instructions
- Reorder machine instructions to avoid dependencies

8 Caches

8.1 Replacement policies

LRU Overwrite the least recently used element.

FIFO Overwrite the oldest element.

LFU Overwrite the least frequently used element.

9 Questions

Questions, you should be able to answer:

1. Which registers does x86 define? How can you address the first 8 bits of a register? What are EFLAGS?
2. Which syntaxi are in use to write x86 assembler? What does the suffix "l" stand for? What is a word? What do the various modes look like syntactically?
3. How is an array stored at hardware level? Explain the following assembler directives:
   - .string
   - .data
   - .space
   - .file

- .rept / .endr
- .long
- .globl
- .text
- .byte
- .equ
- _start

4. Explain the "-a" and "-gstabs" options of GNU assembler ("as" on CLI). What parts of the ASM source code will occur at the *defined* and the *undefined* symbols section.

5. What's the 10-complement of 89, 9 and 2 (each with 2 digits)? What the 2-complement of 15, 7 and 5 (with 4 digits)?

6. What's the difference between (5 bytes) and (1 long with a word) at hardware layer?

7. Describe the procedure of a function call with a stack dump. How are function parameters and local variables assigned? What are the commands push, pop, call and ret for?

8. Describe the following assembler operations:

   - movl
   - addl
   - decl
   - cmpl
   - call
   - ret
   - jnz
   - jz
   - jge
   - js
   - jmp

9. What is each of the following programs doing in the compilation or software development process? readelf, objdump, ar, ranlib, ld, gdb, gcc.

10. Name a command to link several object files to an executable.

11. How can state be stored at hardware level? Explain it using a diagram of a latch and a flip flop.

12. Which software and hardware layers do you know?

13. Describe the fetch and execute algorithm.

14. Describe the dependencies making work with pipelining difficult and discuss possible solutions.

15. What are caches? Which cache size is recommended? Describe the principle of locality. Describe possible replacement policies. What is direct and associative mapping?

16. Describe the idea of a DMA.