Architektura komputerów

Wykład 6

Jan Kazimirski

Architektura x86

Środowisko wykonawcze x86 (32-bit)

- Przestrzeń adresowa
  - Liniowa przestrzeń adresowa do 4 GB
  - Fizyczna przestrzeń adresowa do 64 GB
- Rejestry podstawowe
  - 8 rejestrów ogólnego użytku
  - 6 rejestrów segmentowych
  - rejestr znaczników (EFLAGS)
  - licznik rozkazów (EIP)

Środowisko wykonawcze x86 (32-bit) c.d.

- Rejestry FPU
  - 8 rejestrów danych (80-bitowe)
  - rejestry kontrolne
- Rejestry MMX
  - 8 rejestrów danych 64-bitowych
  - 8 rejestrów danych 128-bitowych

Środowisko wykonawcze x86 (64-bit)

- Przestrzeń adresowa
  - Liniowa przestrzeń adresowa 2^{64} bajtów
  - Fizyczna przestrzeń adresowa 2^{40} bajtów
- Rejestry podstawowe
  - 16 rejestrów ogólnego użytku (64-bitowe)
  - 6 rejestrów segmentowych (64-bitowe)
  - 64-bitowy rejestr znaczników (RFLAGS)
  - 64-bitowy licznik rozkazów (RIP)

Środowisko wykonawcze x86 (64-bit) c.d.

- Rejestry FPU
  - 8 rejestrów danych (80-bitowe)
  - rejestry kontrolne
- Rejestry MMX
  - 8 rejestrów danych 64-bitowych
  - 16 rejestrów danych 128-bitowych

Rejestry ogólnego użytku (32-bit)

- Rejestry: EAX, EBX, ECX, EDX, ESI, EDI, EBP, and ESP.
- Rejestry mogą być dowolnie używane ale niektóre operacje wykorzystują specyficzne rejestry:

EAX - akumulator (obliczenia)
EBX - wskaźnik danych (z rejestrem segmentowym DS)
ECX - licznik pętli
EDX - używany w operacjach we/wy
ESI - wskaźnik źródła danych (przesłania blokowe)
EDI - wskaźnik celu danych (przesłania blokowe)
ESP - wskaźnik stosu
EBP - dodatkowy wskaźnik danych na stosie

Rejestry ogólnego użytku (32-bit) c.d.

- Rejestry 32-bitowe mogą być traktowane jako rejestry 16-bitowe (16 mniej znaczących bitów) o nazwach odpowiednio AX, BX, CX, DX, BP, SI, DI, and SP
- 16-bitowe rejestry AX, BX, CX, DX mogą być traktowane jako pary rejestrów 8-bitowych AH/AL, BH/BL, CH/CL, DH/DL.

Rejestry ogólnego użytku (64-bit)

- Dostępne wszystkie rejestry trybu 32-bit
- Rejestry można też używać jako 64-bitowe. mają wtedy nazwy: RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP.
- Osiem nowych rejestrów, które mogą być używane jako 32-bitowe (R8D – R15D) lub 64-bitowe (R8-R15)
- Istnieją pewne ograniczenia w dostępie do rejestrów w trybach 16- i 8-bitowym

Rejestry segmentowe (32-bit)

- Rejestry 16-bitowe – CS, DS, SS, ES, FS, GS
- Stosowane w modelu pamięci opartym o segmentację:

CS	- segment kodu
DS	- segment danych
SS	- segment stosu
ES, FS, GS	- dodatkowe segmenty danych

Rejestry segmentowe (64-bit)

- Tryb 64-bit zwykle stosuje płaski model pamięci (adresy 64-bitowe).
- Rejestry mapują segmenty na liniową pamięć adresową (mają wartości 0).
- Mogą być wykorzystywane przez niektóre operacje.

Rejestr znaczników

- Rejestr znaczników EFLAGS (32-bit), RFLAGS (64-bit) – przechowuje m.in. flagi wskazujące na rezultat wykonanych operacji. Np.:
  - CF (bit 0) – bit przeniesienia
  - PF (bit 2) – bit parzystości
  - ZF (bit 6) – bit zera
  - SF (bit 7) – bit znaku (1 – minus)
- Inne flagi określają tryby pracy CPU, obsługi przerwań i inne.

Assembler i Linux

Assemblery pod Linuxem

- GAS – Gnu Assembler (składnia AT&T)
- NASM – Netwide Assembler (składnia Intela)
- YASM
- FASM – Flat Assembler
- HLA – High Level Assembly Language

Składnia

- Składnia AT & T
  - Stosowana na platformach UNIX-owych
  - Wcześniej jedyna dostępna dla assemblera GAS
- Składnia Intela
  - Używana przez firmę Intel w dokumentacji architektury x86
  - Używana przez assemblery DOS/Windows i Linux (NASM)

Assembler NASM

- Assembler 80x86 i x86-64.
- Wspiera liczne formaty wyjściowe, m.in. elf32, elf64 (linux), obj, win32, win64 (DOS/Windows)
- Stosuje składnię assemblera firmy Intel
- Wspiera większość obecnych rozszerzeń architektury x86 (m. in. MMX, SSEn)
- Dokumentacja:
  http://www.nasm.us/xdoc/2.09.03/html/nasmdoc0.html

Praca z NASM

- Przygotowanie programu źródłowego
- Kompilacja do pliku obiektowego (binarnego)
- Konsolidacja (linkowanie) do pliku wykonywalnego.
- Uruchomienie programu wykonywalnego

Plik źródłowy

- Zwykły plik tekstowy
- Rozszerzenie .asm lub .s (stosowane przez GNU)
- Dowolny edytor: gedit, kate, vim

Kompilacja

- Kompilacja w środowisku 32-bitowym
  nasm -f elf32 <plik_źródłowy>
- Kompilacja w środowisku 64-bitowym
  nasm -f elf64 <plik_źródłowy>
- Rezultat: plik binarny (obiektowy) - .o

Konsolidacja

- Środowisko natywne
  ld <plik_obj>
- Konsolidacja 32-bitowego pliku binarnego w środowisku 64-bitowym
  ld -m elf_i386 <plik_obj>
- Rezultat: plik wykonywalny o domyślnej nazwie **a.out**.

Flagi kompilatora i konsolidatora

- Przydatne flagi kompilatora:
  - **-f <format>** określa format pliku binarnego
  - **-l <plik>** generuje listing programu
  - **-o <nazwa>** określa nazwę pliku binarnego
- Przydatne flagi konsolidatora
  - **-o <nazwa>** nazwa pliku wykonywalnego

„Hello world”

```
segment .data
msg     db      'Hello World!!',0x0a   ; text to display

segment .text
        global  _start

_start:
        mov     eax, 4                 ; syscall code 4 (write)
        mov     ebx, 1                 ; file descriptor (stdout)
        mov     ecx, msg               ; buffer address
        mov     edx, 14                ; buffer size
        int     0x80                   ; do syscall

        mov     eax, 1                 ; syscall code 1 (exit)
        mov     ebx, 5                 ; exit status
        int     0x80                   ; syscall 1 - exit program
```

NASM - składnia

- Format linii kodu źródłowego:

  etykieta: *instrukcja* *operandy* *; komentarz*

- Większość pól jest opcjonalna (zależnie od instrukcji).
- Znak \ na końcu linii oznacza kontynuację w linii następnej.

NASM – składnia c.d.

- Dodatkowe białe znaki (spacje,tabulacje) w linii są ignorowane.
- Znak : po nazwie etykiety jest opcjonalny.
- Etykieta może składać się z dowolnych znaków, zaczyna się od litery.
- Etykieta o nazwie zastrzeżonej (np. eax) można poprzedzić znakiem $.

NASM – składnia c.d.

- Etykieta zaczynająca się od . jest etykietą lokalną (przypisaną do poprzedniej etykiety globalnej)
- Pole instrukcji może zawierać kod dowolnej instrukcji x86.
- Operandy mogą mieć różne formaty: nazwy rejestrów, adresy, stałe lub wyrażenia
- Pole instrukcji może zawierać pseudo-instrukcje asemblera.

NASM - pseudo-instrukcje

- Deklaracje zainicjowanych danych

  db – dana jedno-bajtowa
  dw – słowo dwu-bajtowe
  dd – słowo 4-bajtowe
  dq – słowo 8-bajtowe

- Przykłady:

```
db    0x55                ; liczba 0x55
db    0x55,0x56,0x57      ; trzy kolejne liczby
db    'a',0x55            ; stała znakowa i liczba
db    'hello',13,10,'$'   ; łańcuch znaków
dw    0x1234              ; słowo (0x34 0x12)
```

NASM - pseudo-instrukcje

- Deklaracje niezainicjowanych obszarów pamięci:
  - **resb** - jeden bajt
  - **resw** - słowo 2-bajtowe
  - **resd** - słowo 4-bajtowe
  - **resq** - słowo 8-bajtowe
- Przykłady:

```
resb    64          ; rezerwuje 64 bajty
resw    1           ; rezerwuje 1 słowo 2-bajtowe
resq    10          ; rezerwuje 10 słów 8-bajtowych
```

NASM - pseudo-instrukcje

- Włączanie binarnych danych z pliku

  incbin „nazwa”

- Aby ominąć pierwsze N bajtów pliku można użyć składni: *incbin „nazwa”,N*
- Aby ominąć N bajtów i wczytać tylko M kolejnych bajtów można użyć: *incbin „nazwa”,N,M*

NASM - pseudo-instrukcje

- Deklaracja stałych – equ.

```
start:    equ     0xFF
```

- Powtarzanie instrukcji – times.

```
zerobuf:        times 64 db 0
                times 100 movsb
```

Adresowanie efektywne

- Adres efektywny to wyrażenie w nawiasach []
- Przykłady:

```
wordvar dw      123
mov     ax,[wordvar]
mov     ax,[wordvar+1]

mov     eax,[ebx*2+ecx+offset]
mov     ax,[bp+di+8]
```

Stałe

- Stałe liczbowe, m.in. dziesiętne, heksadecymalne, binarne. Przykłady: 100, 100d, 0xc8, 11011111b
- Stałe znakowe (rozmiar do 8 bajtów) – otoczone znakami ' lub „
- Stałe tekstowe – j.w. ale bez ograniczenia.
- Inne (liczby zmiennoprzecinkowe, kodowane BCD itp)

Wyrażenia

- Składnia wyrażeń i operatory odpowiadają składni C
- Znak $ oznacza aktualną pozycję w kodzie.
- Operatory:
  - logiczne (|,&,^,!)
  - przesunięcia (>>,<<)
  - arytmetyczne (+,-,*,/,%)
  - inne ...

Makrodefinicje

- Preprocesor NASM-a zawiera bardzo rozbudowany preprocesor. Umożliwia on m.in.
  - definiowanie prostych makro-definicji
  - makro-definicji z parametrami
  - włączanie plików
  - kompilacja warunkowa
  - makro-definicje obsługujące łańcuchy znakowe
  - wiele innych ...

Makrodefinicje - przykłady

```
%define ctrl    0x1F &          ; proste makro
%define foo bar                 ; zastąpienie nazwy
%define b(x)    2*x             ; makro z parametrem
%strlen charcnt 'my string'     ; długość tekstu
%include "macros.mac"           ; włączenie pliku

%ifndef MACROS_MAC              ; włączanie z zabezpieczeniem
    %define MACROS_MAC          ;  przed wielokrotnym włączeniem
    (...)                       ;  za pomocą kompilacji warunkowej
%endif
```

Dyrektywy assemblera

- **bits** – typ generowanego kodu (16-, 32-bity itd.)
- **default** – zmienia domyślne ustawienia NASM-a
- **segment** – definicja segmentów programu
- **absolute** – generowanie adresów absolutnych
- **extern** – importowanie symboli z innych modułów
- **global** – eksportowanie symboli do innych modułów
- **common** – definiowanie wspólnego obszaru danych globalnych
- **cpu** – typ procesora (lista rozkazów)
- **float** – opcje FPU