Organizacja i architektura komputerów

Piotr Patronik

25 lutego 2016

Kontakt

dr inż. Piotr Patronik
Politechnika Wrocławska
Wydział Elektroniki
Katedra Informatyki Technicznej

- tel. +48 713202759
- skype: piotrpatronik
- email: piotr.patronik@pwr.wroc.pl
- biuro: pok. 220 C-3
- `http://zak.pwr.wroc.pl/pepe`
- konsultacje: wt. 15-17, cz. 11-13
- (lista obecności)

Plan kursu

1. Literatura, zaliczenie, sformułowanie problematyki
2. Języki maszynowe, architektura listy rozkazów
3. Reprezentacja i typy danych, tryby adresowania
4. Sterowanie wykonaniem programu
   → Warunki i rozgałęzienia, funkcje, pętle
5. Programowanie w asemblerze
   → Tworzenie i uruchamianie programów
6. Organizacja i hierarchia pamięci
   → Metody przyspieszania dostępu
7. Zasada lokalności - pamięć podręczna
   → Organizacja, problem spójności, sterowniki i magistrale pamięci
8. Zarządzanie pamięcią
   → Ochrona pamięci, segmentacja i pamięć wirtualna, stronicowanie
9. Przerwania wewnętrzne i zewnętrzne
   → Wyjątki i ich obsługa
10. Przetwarzanie potokowe
    → Konflikty i ich usuwanie
11. Współpraca wielu jednostek wykonawczych
    → Algorytm Tomasulo
12. Interfejsy i magistrale
    → Przestrzeń i obsługa we/wy
13. Kody korekcyjne i detekcyjne w przetwarzaniu danych
14. Niezawodność i wiarygodność
    → Systemów i obliczeń
15. Podsumowanie

Literatura

- J. Biernat, Architektura Komputerów
- R. Stallings, Organizacja i architektura systemu komputerowego
- J. Null, J. Lobur,
  Struktura organizacyjna i architektura systemów komputerowych

- J. Hennessy, D. Patterson, Computer Architecture:
  A Quantitive Approach
- D. Patterson, J. Hennessy, Computer Organization and Design:
  The Hardware/Software Interface
- R. Stallings, Computer Organization and Architecture:
  Designing for Performance
- W. Gilreath, P. Laplante, Computer Architecture:
  A Minimalist Perspective

- J. Bartlett, Programming from the ground up
- Intel Archtecture Developers' manual
- GNU binutils
- GDB (GNU Debugger)

Literatura, cd - źródła informacji, narzędzia

- `http://scholar.google.com`
- IEEE: `http://ieeexplore.ieee.org`
  (z adresów PWr - ict-stud itp.)
- IEEE Standard 1003.1 (POSIX.1)

Zasady zaliczenia

- Egzamin
  1. Warunki konieczne: zaliczenie projektu i laboratorium
  2. Test otwarty
  3. 5+1 pytań, 25 pkt
  4. 40 min
  5. Zaliczenie od 13 pkt
  6. Ocena celująca: 24+ pkt
  7. Uwzględnienie ocen z projektu i laboratorium

Projekt

- Temat
- Cel projektu
  (niedo)określony przez prowadzącego
- Deliverables (kod/wyniki badań)
- Sprawozdanie (wnioski!)
- Sprawozdanie – szablon
- Konsultacje projektowe

Architektura i organizacja komputera

- Architektura komputera
  specyfikacja funkcjonalnych cech komputera opisanych listą rozkazów i wskazanie obiektu ich oddziaływania (architektura listy rozkazów - ISA)
  - Rejestry, pamięć
  - Przestrzeń wejścia/wyjścia
  - Urządzenia: drukarka, dalekopis, dziurkarka kart...
  - ALU, FPU, SIMD, GPU...
  - Motorola, PowerPC, x86, ARM, MIPS, RISCV...
  - Magistrale
  - Wirtualizacja
  - Rozkazy specjalizowane (FPU, h264, AES, DOM...)

Organizacja komputera

- Organizacja komputera – struktura logiczna *odwzorowująca cechy funkcjonalne i nadająca kształt operacyjny (architektura układów – HSA)*
  - Układy arytmetyczne i logiczne (LUC!)
  - Sumatory (PPA), matryce mnożące
  - Standardy kodowania danych i zapisu liczb (U2, 754, RNS...)
  - Maszyny stanów, mikroprogramy
  - Potokowanie i potoki programowalne

Poziom fizyczny

- Wykonanie (technologia)
  - Układy mechaniczne
    (liczydło, katarynka, układy krzywkowe. . .)
  - Układy elektromechaniczne (przekaźniki)
  - Lampy elektronowe
  - Tranzystory (germanowe/krzemowe)
  - Układy scalone
    (TTL, NMOS, CMOS, ThinFET, FinFET, 3D. . .)
    10μm (1971) $\rightarrow$ 5nm (2021)
  - QCA, RTD. . .

(ITRS – International Technology Roadmap for Semiconductors)

Model programowy procesora

- Rejestry (plik rejestrów)
- Rozkazy (lista rozkazów)
- Tryby adresowania
- Pamięć (organizacja pamięci)
- Wejście/wyjście (interfejs użytkownika)
- Funkcje systemu operacyjnego

- Model programowy vs model obiektowy

Narzędzia podstawowe

- System operacyjny i aplikacje (iOS/Android/Win/Linux/Unix/Arduino)
- Konsola i linia poleceń
- Interfejs użytkownika, komendy i (anty)intuicje
- Przestrzeń pojęć, obrazów, abstrakcji i algorytmów

- Zapis kodu programu (Scratch vs plik tekstowy)
- Źródło programu a postać wykonywalna

- Powłoka systemu operacyjnego – bash
- Edytor tekstu (notatnik) – vim
- Asembler („kompilator”) – as
- Konsolidator – ld
- Debugger (program uruchamiający) – gdb

Poziomy maszynowe i języki opisu

Poziom programowy

- aplikacja
- język makropoleceń (funkcje biblioteczne)
- język algorytmiczny (opis/programowanie algorytmów)
- język asemblerowy (programowanie maszyny)
- system operacyjny (funkcje systemu operacyjnego)
- poziom abstrakcji sprzętu/oprogramowanie wbudowane/sterowniki

Poziom sprzętowy

- dekodowanie rozkazów procesora
- sterowanie bloków i ścieżek danych
- struktura logiczna i architektura (automaty, ALU, rejestry)
- układ cyfrowy, bramki, tranzystory...
- topografia układu

Program vs algorytm

- OiAK/AK2 a PTM
- Implementacja algorytmu
- Konstrukcje algorytmiczne: pętle, warunki...
- Złożoność problemu a krzywa uczenia

- Zrozumienie kodu (analiza), a
  napisanie go samodzielnie (synteza)
- Pisanie samodzielne a przepisywanie
- Programowanie a znajomość architektury (i organizacji...)

Najprostszy program (w C)

endme.c – zakończenie procesu

```
#include <unistd.h>

int main()
{
_exit(0);
}
```

Najprostszy program - kod

endme.s

```
# vim: syntax=gas

.text
.global _start
_start:
mov $1, %eax
mov $42, %ebx
int $0x80
```

(:set syntax=gas dla VIM-a; w .vimrc trzeba jeszcze dodać set modeline)

Architektury obecnych systemów

Architektura 32-bitowa (dla niej prezentuję przykłady)

```
$ uname -m
i686
$
```

Architektura 64-bitowa

```
$ uname -m
x86_64
$
```

Inny model programowy: rejestry, funkcje systemowe...

Najprostszy program - „kompilacja”

Asemblacja

```
$ as -o endme.o endme.s
```

Konsolidacja

```
$ ld -o endme endme.o
```

(w środowisku VS w/w kroki są wykonywane automatycznie po F5)

Najprostszy program - uruchomienie

Uruchomienie

```
$ ./endme
$ echo $?
42
$
```

- Krok automatyczny w środowisku programistycznym
- Można też kliknąć ikonę w oknie

Najprostszy program - sesja debuggera

Uruchomienie GDB

```
$ gdb -q ./endme
Reading symbols from /home/pepe/Documents/ak2-wyklady/wy
(gdb) r
Starting program: /home/pepe/Documents/ak2-wyklady/wykla

Program exited with code 052.
(gdb) q
$
```

- Sukces: 052 to 42 w zapisie ósemkowym

Model programowy - program w pamięci

- Pamięć to tablica liniowa
- Rozkazy są umieszczone w kolejnych adresach
- Każdy rozkaz to ciąg bajtów – słowo rozkazowe
  $\longrightarrow$ stała lub zmienna długość słowa rozkazowego
- Program Counter (PC) lub Instruction Pointer (IP)

Najprostszy program - sesja debuggera cd

Zrzut pamięci programu

```
(gdb) disassemble *_start
Dump of assembler code for function _start:
   0x08048054 <+0>:     mov    $0x1,%eax
   0x08048059 <+5>:     mov    $0x2a,%ebx
   0x0804805e <+10>:    int    $0x80
End of assembler dump.
(gdb)
```

Słowa rozkazowe w pamięci

```
(gdb) disassemble /r *_start
   0x08048054 <+0>:     b8 01 00 00 00 mov    $0x1,%eax
   0x08048059 <+5>:     bb 2a 00 00 00 mov    $0x2a,%ebx
   0x0804805e <+10>:    cd 80  int    $0x80
End of assembler dump.
(gdb)
```

- Tryb adresowania natychmiastowego – stała w kodzie rozkazu

Najprostszy program - sesja debuggera cd

Punkt zatrzymania programu (breakpoint)
To **NIE** zadziała:

```
(gdb) b *_start
```

Pierwszy punkt zatrzymania na kolejnym rozkazie

```
(gdb) b *_start+5
(gdb) info breakpoints
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x08048059 <_start+5>
(gdb) r
Starting program: /home/pepe/Documents/ak2-wyklady/wykla

Breakpoint 1, 0x08048059 in _start ()
(gdb) cont
Continuing.

Program exited with code 052.
(gdb)
```

Pytania? Uwagi?