Poznámky z principů PC

Sonic the hedgehog

Autor: Milan Veselý

Odkazy na přednášky jsou tajný, takže je nedám sem

Ale na disku je ve stejný složce pdfko s odkazama

1. Přednáška –Úvod (Harvard, druhy přenosů)

Harvardská architektura

CPU (procesor) – Nejdůležitější součást počítače, která vykonává instrukce

Kódová paměť – Procesor z ní čte instrukce

Datová paměť – Uchovává pracovní data

Periferie (I/O)

Historie: Babbage, Lovelace, …

Kódování informace

Analogový přenos

Jak to zakódovat? Očividně nemůžeme způsobem $1 = 1V$.

Můžeme tedy přiřadit násobky, ale i to je špatné kvůli odporu, teplotě, …

Digitální přenos

Zavedeme si bity a tedy budou pouze dvě hodnoty

Šedá zóna je rozptyl napětí, kde to není chápáno ani jako 1 ani 0

Paralelní přenos

Více bitů zároveň přes několik vodičů

Sériový přenos

Posíláme bity za sebou – zakreslíme časovým diagramem

Pojem *rising edge* v diagramu u změny napětí

1. Měření napětí vůči zemi

   Napětí je relativní veličina a tedy je potřeba i referenční vodič

2. Měření rozdílu – Diferenciální přenos

   To se dá vyřešit tím, že použijeme dva datové kabely a budeme do obou posílat protichůdné bity

   Používá ho například USB (dva piny datové, dva na napájení)

Musí být jasně daná délka bitů (transfer rate v baud / s)

LSb a MSb first

Zda posíláme prvně ten nejmenší (least significant) bit nebo ten největší (most significant) bit

2. Přednáška – Detaily přenosu

Přenos

Synchronizace hodin

Hodiny přenosu se ale časem rozejdou

Řešení 1 – vymyšlení nového stavu (3 stavová logika)

- Jak vůbec poznáme začátek přenosu?
  - Zavedeme si idle stav
  - Ten realizujeme buď fyzicky – floating state (Hi-Z)
    - To ale není moc praktické
  - Nebo si ho definujeme na základě toho co už máme
    - Na začátku je idle (0) a přenos začne až po strat bitu
    - Hodiny se synchronizují na základě rising edge
- Řešení synchronizace hodin
  - Omezíme přenos na n bitů (IRL 8 bitů)
    - Zavedeme tedy 1 byte (1B) nebo-li octet = 8 bitů (8b)
  - Velký overhead (8 datových bitů potřebuje start bit a stop bit)
    - A tedy 800 datových bitů na 1000 baudů
    - Právě počet bitů dat se většinou udává u rychlosti b/s
- Používá ho například RS-232 linka

Řešení 2 – hodinový signál

- Produkuje ho buď jedno ze zařízení nebo nějaké externí
- Nový vodič se signálem kdy číst data
- Musíme se nějak dohodnout kterou hranu detekovat (rising, falling)
  - Kvůli tomu ale bude přenos poloviční rychlosti
- Double data rate tento overhead řeší
  - Detekuje se klesající i stoupající hrana
- Stále ale potřebuji start a stop bit
- Například I^2C linka

Řešení 3 – clock recovery

- Budeme signál průběžně srovnávat
- Problém, když bude dlouho monotónní signál
- To se vyřeší převodem dat z 8b na vyšší (např. 10b)
  - Vybírají se jen hezké pomocí tabulky
- Např. USB

Typy přenosu

Simplexní přenos

Jednosměrný přenos (pevně vysílající a přijímací strana)

Half-duplex

Zařízení se v odesílání střídají

Full duplex

Dvě nezávislé simplexní linky

Např. RS-232 (dvě linky a jedna zem)

Má ale více než 3 piny – *Out of band signály*

Také nazývané *význačné*

Signály ano/ne (1/0) – pozor na inverzní logiku

Komunikační protokol

Dohoda o tom jak přenos vypadá (Jaký mají bity význam)

Řadič

Přizpůsobuje komunikaci zařízení s procesorem

Registry (malá pomocná paměť)

Data registr – buffer (dočasně uchová data ke zpracování)

Konfigurační registr – nastavení rychlosti, parity, …

Stavový registr – stav načtení celého bytu

3. Přenáška – Komunikační protokol, binární a šestnáctková soustava

Připojení myši

Prvně všechny out of band signály nastavím na 0 a počkám

Tím myš přijde o napájení a vypne se

Při zapnutí se vypíšou nějaká inicializační data

Blokující x neblokující funkce

Čekáme než přijde určité množství dat – blokování

Pokud nepřijdou do určité doby, jdeme dál

Přidáme timeout

Pozn. v binárce indexujeme tak, že mocnina odpovídá indexu

Jak zjistit, který bit je nastavený

Použijeme bitovou masku (přiANDujeme určité číslo…)

Číslo s nulami na všech místech kromě toho, které nás zajímá

Binární a šestnáctková soustava

Úvod se mi nechtěl psát – jak se převádí, proč se používá, jak značíme …

Bitové operace

Standartní logické operace

Binární operace

AND (&), OR (|), XOR (^)

Unární operace

NOT (~)

Bitové posuny

Posun doleva – SHL (<<) Posun směrem k MSb!

Posun doprava – SHR (>>) …

Bitové rotace

ROL a ROR

Bity se při vysunutí nezahodí, ale vrátí na začátek

4. Přednáška – Záporná čísla v binárce a čísla v Pythonu

Záporná čísla

Signed integer x unsigned integer

První reprezentace

Přidáme na začátek znaménkový bit (0 – plus, 1 – minus)

Špatně se to ale porovná, menší číslo má totiž větší hodnotu

Jedničkový doplněk

Všechna záporná čísla navíc bitově zneguju

Problém je s tím, že existují dvě různé nuly

Dvojkový doplněk

Vyřeším předchozí problém tím, že k negaci přičtu 1

Zvětší se nám také rozsah -2^{n-1} až $2^{n-1}-1$

Pozn. bit_length nevrací overhead čísla (znaménkový bit, …)

Čísla v Pythonu

Python mění délku čísel podle toho jak se mu to hodí

Pokud to chceme zafixovat, tak použijeme numpy

Truncation – zkrácení čísla

Problém s useknutím záporného čísla…

(Zero/signed) extension – …

5. Přednáška – Master a Slave, sběrnice

Operační paměť

Data memory počítače

Master a slave

Master (řídící) dává požadavky a Slave (řízený) vykonává požadavky

CPU je obvykle master

Point to point linka

Z jednoho zařízení do druhého (jeden master, jeden slave)

Multidrop linka/bus/sběrnice

Sběrnice actually znamená něco jiného

Více slavů na jedné lince

Rozlišujeme je adresami (rozsah je adresový prostor)

Detekce zařízení na lince

Pull-up nebo pull-down resistor

I²C sběrnice

Je *multimaster* (může mít více masterů) *např USB je single master*

SCL (serial clock) a *SDA* (serial data)

Pokud nesíthá, tak může dělat clock stretching

Start condition, stop condiotion, IDLE stav, …

8 datových bitů v MSb first

9. *ACK* bit – potvrzení

Payload – data, která nás zajímají

Příklad na ambient light sensor

ADC/analog digital converter – převodník z analogu na digitální

Bus interface/rozhraní – připojení ke komunikační lince

Počítací registr (read only), Příkazový registr (read only)

(nemusíme specifikovat s jakým z těch dvou chceme komunikovat)

Protože má pevně danou adresu, tak na sběrnici může být jen jedno

Více bytová hodnota – MSB/LSB first (specifikuje si výrobce)

6. Přednáška – RAM

Paměť

Který konkrétní byte nás zajímá? – adresa bytu

Paměťový adresový prostor – rozsah adres v paměti

Jeho rozsah nemusí odpovídat množství paměti

Většinou to je nejbližší vyšší násobek 8 (bývá ale i větší např pro škálování)

1kB = 1KiB = 1024B (1000B je nepraktické a používají to jen výrobci disků)

RAM - Random access memory

Znamená to především to, že přístup ke každému místu trvá stejně dlouho

Existuje ale i druhý pohled, že stačí, že může přistupovat kamkoliv…

V reálu je ale pro ni nejlepší přístup *sekvenční* (k rostoucím adresám)

Je volatile (po vypnutí ztratí obsah) a read/write

SRAM

Každý bit je ze 4-6 tranzistorů

Relativně malá paměť, ale velmi rychlá – stovky GB/s (sekvenčně)

Access time (nalezení konkrétního bitu) – méně než nano sekunda

DRAM

Bit je z 1 tranzistoru a 1 kondenzátoru

Jednotky až desítky GB

Zhruba 10x pomalejší a *access time* v nanosekundách

Velmi rychle zapomíná a je potřeba refresh

Slovo – množství bitů v jednom přenosu

(Špatná definice je slovo = 16b, potom jsou i double word (32b), quad word, …)

n-bitové zařízení – *n* je velikost slova se kterou pracuje

Overhead přenosu na PCF8570

Při 100 000Hz dostaneme reálně jen $100000/(3 \cdot 9)$Bps (3708Bps)

Musíme započítat ACK bity a dva byty na adresu

Zlepšení je *burst přenos* (automaticky inkrementujeme adresu)

$(x - (2 \cdot 9))/9$ a tedy 11109Bps

Btw čtení je dvakrát rychlejší než zápis

6,5. Přednáška – Registrový adresový prostor

Registrový adresový prostor

Pokud má zařízení více registrů máme obdobné adresy jako u paměti

7. Přednáška – Procesor (Kompilátor x interpret, architektura, strojový kód)

Procesor

Instrukce

Obvyklé jazyky jsou na procesor příliš složité

Proces tedy zpracovává *instrukce* – posloupnost *n*-bytů

Z výroby má danou *instrukční sadu*, kterou zná

Strojový kód je více instrukcí za sebou

Program counter/Instruction pointer – ukazatel aktuální adresy

Více bytové instrukci adresuji tou nejnižší adresou

Potřebuje oadresovat celou code memory

Instrukce je *Opcode* – druh instrukce (sčítání, odečítání, …) + *argumenty*

Zpět k obvyklým jazykům, ty dělíme na dva druhy:

Komplikované

Kompilátor ve strojovém kódu přeloží text na strojový kód

Pak to za pomocí *magie* nakopíruje z data memory do code memory

Interpretované

Místo kompilátoru používá interpret a ten se chová podle kódu

Je to ale pomalejší

Adresa

Adresy proměnných zařizuje kompilátor/interpret

Jak ale data ukládat? Od začátku nebo od konce?

Endianita

Little endian

Least significant byte na nejnižší adrese

Big endian

Tak jak bych to napsal normálně

Největší (most significant) na začátku

Každý procesor má svoji endianitu z výroby

Dnes je většina little endian

Problém při komunikaci pokud jiné zařízení používá big end

Architektura

Pro kódovou paměť musíme zvolit nonvolatile paměť, jinak by se při zapnutí nevědělo co se má vykonávat

Důvod proč i data memory není nonvolatile je rychlost a životnost

Spousta procesorů nemá *magii* a tedy přesun dat do code memory musíme vyřešit nějak jinak

Zavádíme Von Neumannovskou architekturu

Jedna paměť pro data a kód

Musíme ale nějak nakopírovat data před spuštěním

Příklady procesorů

Historie počítačů…

6502, Intel 8088

Instrukce v praxi

Běžné operace

V procesoru je problém s operací a = b *op* c a používáme a = a *op* c

Jak tedy udělat a = b + c ? Přes pomocný registr

(načteme z registru, přičteme, uložíme do registr, z registru přiřadíme)

Assembler

Protože jsou machine code nepřehledný, tak se zavádí Assembler

Přehled instrukcí

6502	**Machine code**	**Assembler**
Do nothing	$EA	NOP
Jump to	$4C XX_0 XX_1	JMP $$XX_1$ XX_0
Load constant into A	$A9 XX_0	LDA #$$XX_0$
Load value from adress …	$AD XX_0 XX_1	LDA $$XX_1$ XX_0

Pro registr X je to $A2, $AE a LDX

Potom tam jsou instrukce pro store STA, STX, …

A instrukce pro převod mezi registry TAX, TXA, TAY …

Intel x86	**Machine code**	**Assembler**
Do nothing	$90	NOP
Jump to	$E9 XX_0 XX_1 XX_2 XX_3	JMP $$XX_1$ XX_0

8. přednáška – Procesor 2 (registry a operace)

Příznakové (flag) registry

Příznak je jednobitová informace

zero – jestli poslední výsledek byla nula

sign – znaménko poslední operace

carry – přenos z poslední operace

Instrukce CLC (clear carry), SEC (set carry)

Akumulátorová architektura - všechny aritmetické operace z registru A

Bitové operace 6502

ORA, AND, EOR (NOT není, musíme EORovat samé jedničky)

Pro více bytové hodnoty to provádíme postupně

ASL, LSR, ROL, ROR

Aritmetické operace

ADC (add with carry) pokud chceme sčítat více bytové hodnoty

Carry musí před tím nastavit na 0

Je velmi důležitá endianita

Odečítání můžeme vyřešit přičtením záporného

Znegujeme a přičteme 1 (A + NOT(B) + 1)

Lepší řešení je ale SBB (subtract with borrow)

Půjčování si hondoty z vyššího řádu

Je ale možný i SBC (substract with carry)

$$A - X - B \rightarrow A - X - (1 - C) + 256 \rightarrow A + NOT(X) + C$$

x86

32-bitový

Mají i instrukce v menší přesnosti

Více registrů a obecné registry (můžeme s nimi dělat co chceme)

Snadno můžeme řešit komplexní výrazy

Instrukce MOV, ADD, ADC, …

9. Přednáška – Rychlost operací, Efektivita Pythonu, Fixed point

Rychlost operací

Jednotka je 1 takt procesoru

Dnes všechny základní instrukce trvají jeden takt

Load a store jsou pomalé, kvůli pomalejší paměti

Efektivita Pythonu

- Proměnné obsahují adresu (je to pointer) (4B nebo 8B)
  - Na adrese je ta hodnota (min 4B), její velikost (4B), garbage collector počítadlo (4B) a typ proměnné (4B)
  - A tedy 4B hodnota zabere 24B
  - Časté hodnoty (od -5 do 256) jsou naštěstí někde v tabulce
- Před každou operací se to musí prověřit (a navíc je interpretovaný)
  - Python je tedy většinou ve výsledku 100x pomalejší
- Má to ale i výhody – neřešíme overflow a underflow
- Pozor reálně je ve 32b jenom 30b hodnot (znaménko a carry)

Násobení

- Mnoho procesorů umí násobit i dělit, ty jednodušší ale často ne
- Násobení je na 10 taktů a dělení na 10-100 taktů
- Pokud to není podporované, tak to musíme vyřešit softwarově
  - Využijeme shiftování jako dělení a násobení 2
  - Jak to bude fungovat se znaménky?
    - SHL funguje pro dostatečně malá čísla
    - SHR nefunguje vůbec
      - Zavedeme SAR a ten rozšiřuje znaménkově
      - To se chová divně na sudých a lichých číslech

Reprezentace desetinných čísel

Fixed point

- Pevný počet bitů na každou část
- Není to ale příliš používané, ačkoliv se snadno provádí aritmetické operace

10. Přednáška – Floating point, ROM paměť

- Musíme ale trochu opravit násobení a dělení
- Často fixed point není v programovacích jazycích, ale můžeme to udělat sami

Floating point

- Využijeme vědeckou notaci – mantisa a exponent
  - Běžně je mantisa na na least significant, A na MS je znaménko
- Reprezentace se skrytou 1 (předpokládáme, že neexistuje 0)
- Jak napsat exponent? Nepoužíváme dvojkový doplněk, ale bias (s posunem)
  - Nejmenší číslo namapuje do nuly, … (taky se hodí říct posun o kolik)

Pokud zařízení HW nepodporuje floating point, tak spíše použijeme SW fixed

Denormalizace čísel – pousn tak aby čísla odpovídala a dala se sečíst

Hodně čísel nelze reprezentovat jako floating point

u floatů se nepoužívá rovná se, ale porovnává se rozdíl od epsilon

Dost často to ale můžeme vyřešit celočíselným počítáním v mm, haléřích, …

Standard IEEE

Definuje 32b a 64b floating point čísla

32b – znaménko 1b, exponent 8b, mantisa 23b

64b – znaménko 1b, exponent 11b, mantisa 52b

Takhle je float v Pythonu

V C# je rozdíl mezi float a double

Minimální exponent a nulová mantisa je 0

Problém, že máme dvě nuly, ale to v podstatě nevadí

Maximální exponent a samé 1 je podle znaménka $+\infty/-\infty$

∞/∞ je *nan* (not a number) – reprezentace max exponent a cokoliv

Paměť ROM

Read only memory

Ale hlavní vlastností ROM je, že jsou non volatile

PROM

Programmable ROM

Sám si ji naprogramuji tak, že spálím diody

EPROM

Erasable PROM

Dá se vymazat vystavením UV zářízení

11. Přednáška – ROM paměť, HDD, Soubory

EEPROM

Electrically EPROM

NVRAM (non-volatile RAM, ale RAM znamená, že je r/w)

Reálně se zapisování po nějaké době rozbije

Flash paměť

Paměť rozdělená do bloků (rychlejší na přístup k větším datům, ale …)

Rychlost 10-100 MBps

GPIO

General purpose i/o (slouží pro vstup a výstup)

Hodí se nám u mikrokontrolerů (např. v pračce)

Mimo to budou v mikrokontrolerech i ADC a DAC

Permanentní datové úložiště

Nahraje se konfigurace, pro případ, že umře napájení

HDD

Informace se ukládají magneticky a dlouho vydrží

Disk je rozdělen na koncentrické kružnice (*stopy*)

Dále je rozdělen na *sektory* (Původně měly 512B a nyní mají 4kB)

Plotny jsou nad sebou a každá stran plotny má svoji *hlavičku*

Cylindr – stopu v plotnách nad sebou

Hlavička se posouvá do strany a disk se otáčí

Důležitým parametrem jsou otáčky (*RPM*) cca. 5400-15000

Vhodné přístupy

Sekvenční – nejlepší přístup

Náhodný z jedné stopy – průměrný

Sekvenční pozpátku – vždy se musí počkat až se skoro celý otočí

Náhodné z více stop – úplně nejhorší, protože se musí pohnout hlava

Z toho se skládá adresa (*C/H/S*), ale to se ještě stejně pak převádí

Nejvzdálenější stopy mají ještě "rozpůlené" sektory a jsou tedy rychlejší

Proto se disk zaplňuje a čísluje z vnějšku

CD/DVD/BluRay

Jsou optické – kratší životnost

Pomalejší rychlost než HDD a +100ms přístupová doba

Má jednu spirální stopu a ta je opět rozdělná na 2kB sektory

Číslované od vnitřku (*LBA* - linear block adressing)

SSD disk

Flash paměť s řadičem

Řadiče úložiště

Adresový, command (R/W), buffer, data (počet a velikost sektorů), info

Pro všechny zařízení se navíc používat stejný protokol s LBA

Soubory

Adresování

Offset – posun od začátku sektoru (adresa je absolutní a tohle je relativní)

Base address – adresa od které počítáme

Metada

Data o souboru (kde se nachází,…) na začátku disku

V matedatech je seznamen všech sektorů, kde se nachází, protože

Fragmentované soubory – pokud soubor je rozkouskovaný po disku

Na jeden sektor můžeme uložit pouze jeden soubor

File systém – způsob zápisu metadat

Python

Existuje v něm typ bytes (s klasický overheadem)

Operační systém…

Po open si nakešujeme metadata

```
f = open("jump","r")
```

`"rb","wb"` je v bytech

`seek` nastavení offset

```
f.read(64 )#počet bytů
```

Hexviever

…

Disk Image

Zkopíruju celé sektory jednoho disku do nového disku

Dobré na zkoumání a zálohování disku

12. Přednáška – Reprezentace obrazu a textu

Reprezentace obrazu

Bitmapa, obdélník s hodnotami (*pixely*)

Pixel

Množství fotonů na jednom místě nějak je namapujeme na hodnoty

Bitová hloubka (1b – bitmapa, 8b – hodnoty jsou 1 až 255)

HDR vznikne použitím reálných čísel

Různé bravy mají různé frekvence – toho využijeme při záznamu

Složíme to z R/G/B (16b hloubka (5b,6b,5b), 24b – Truecolor)

Lepší by bylo mít 32b než 24b a proto budeme mít alfa kanál

Jak je ale budeme pixely ukládat? V LE a tedy BGR(A) do jedné proměnné

Formát souboru BMP

Little Endian

Hlavička (*magic constanta* – posloupnost bytů, která říká formát)

Data – bitmapa

Ukládá pozpátku a výška je záporné číslo

Reprezentace textu

Musíme zavést *kódování* (kód = číslo, které reprezentuje znak)

Grafém – jeden nakreslený znak

Mapování na binární reprezentaci (s pevnou/proměnou délkou)

Znaky se ukládají ve stejném pořadí jako se čtou/píšou

ASCII standard

7bitová reprezentace

Písmena A,…,Z,a,…,z,0,…9

Kódování pro češtinu

ISO 8592 – používá ho Linux

852 – používal ho MS DOS

Win 1250 – Windows

Unicode

Je v něm všechno, ale nemá určenou binární reprezentaci

To až **UTF-32** a to je jak v LE tak v BE

12.5 Přednáška – text (kódování a rasterizace), vysvětlení magie (DRAM, PCIe, zapínání)

Výhoda i nevýhoda, že všechny znaky jsou stejně dlouhé

0 až 127 odpovídají ASCII

128 až $FFFF běžné znaky

$10000 až $10FFFF nějaký random

UCS-2 (má pevně 2B) je hloupost, přijdeme o znaky

UTF-16 je lepší, protože může mít 2B i 4B

Abychom zjistili délku souboru, tak to musíme spočítat

Jak se ale pozná který je který? Surrogate – využijeme zakázané dvoubytové znaky a 4B jimi budou začínat

Pak to ještě musíme převést na normální

UTF-8 – nejlepší

Můžeme mít 1B, 2B, 3B i 4B

Zase nějak dáme bity na začátek

Chápe se to vždy jako posloupnost a proto neřešíme endianitu

Pro Windows je ale běžnější LE UTF-16

Rasterizace textu

Víme sice už co to je za znaky, ale teď je musíme nakreslit

Potřebujeme také zalamovat řádky

Carriage returne a line feed (CR + LF)

V ASCII to je 13 a 10 neboli D a A

Windows CR LF a MAC a UNIX LF

Python používá na print to co aktuální systém

Většina programovacích jazyků používá Unicode (UTF-8 nebo UTF-16)

Měli bychom ale v programování specifikovat jaký encoding chceme na open

Text a čísla v jedné posloupnosti bytů

Vlastní formát. Buď s pevnou délkou a nebo dáme délku textu v B

V Numpy je pěkná funkce .tobytes() (je to v endinaitě procesoru)

Obrácené je frombuffer(x, typ)

A str.encode("…") a jeho protějškem str.decode(…)

BTW Bytes je immutable, alternativa je bytearray

Dovysvětlení magie na závěr

Bridge/HBA – řadič nějaké další sběrnice

DRAM paměť

Refresh děláme, tak že připojíme DRAM přes řadič, který to zařídí

Díky řadiči můžeme i připojit více pamětí (v dalších pamětích to mapovat)

Zároveň může být různě velký adresový prostor

HCI (Host controller interface)

Jak komunikovat se zařízením a jeho registry

Proximity sensor a jak číst z více registrů… už bylo v 6.5

Systémová sběrnice (system bus)

Např. PCI express

Znamená to, že připojíme řadič paměti na sběrnici jako ostatní zařízení

Memory read a memory write pakety na které reaguje jen paměť

Je tam jen adresa v paměti a ne zařízení (v MRd je i adresa CPU)

Řadič paměti (memory controller) vykoná požadavek a vrátí CpID

Jak se připojit na ostatní zařízení?

Memory mapped I/O

Pro I/O použijeme volné adresy v paměti

Řeší to i kopírování z řadiče do paměti

Co vykonat při zapnutí počítače

Připojíme non-volatile paměť s firmwarem počítače

A každý procesor má CPU startup vector

Firmware ROM

1. Test a konfigurace hardware (přidělení adres zařízením)
   Plug and play
2. Najít užitečný software
   Option ROM
   Bootování z pevného disku
3. Základní funkce pro bootování – klávesnice, čtení sektoru

Dnes to nahraje kernel operačního systému

FS, vstup (myš, klávesnice), výstup (rasterizace, …), programy