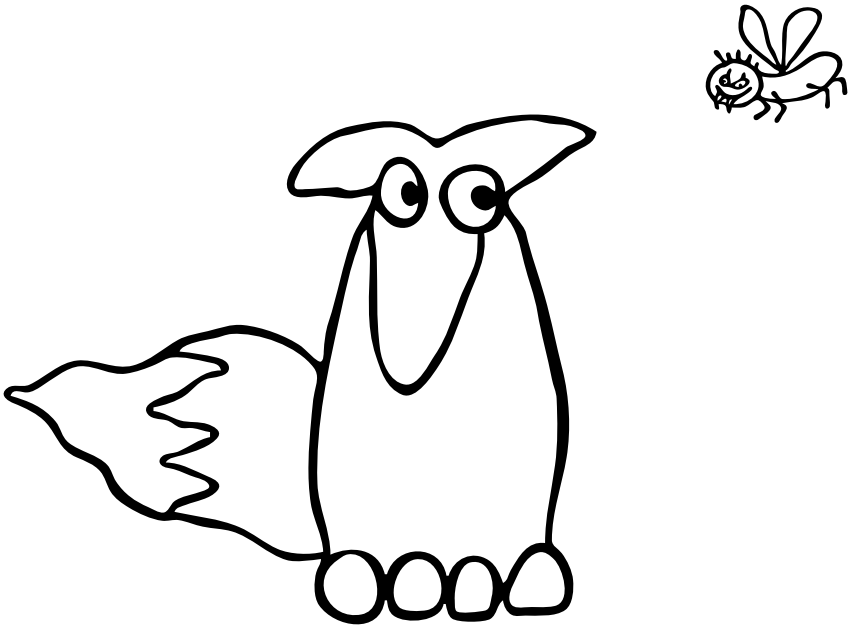
Studentský matematicko-fyzikální časopis

M&M

Ročník XXIII / Číslo 5



Časopis M&M a stejnojmenný korespondenční seminář je určen pro studenty středních škol, kteří se zajímají o matematiku, fyziku či informatiku. Během školního roku dostávají řešitelé zdarma čísla se zadáním úloh a témat k přemýšlení. Svá řešení odesílají k nám do redakce. My jejich příspěvky opravíme, obodujeme a pošleme zpět. Nejzajímavější řešení otiskujeme.

Milí řešitelé,

máme pro vás další číslo časopisu M&M. Naleznete v něm řešení třetí série a také další díl seriálu o assembleru, včetně úloh na procvičení, za které také můžete získat body. Sedmého dubna se bude konat matematická soutěž Náboj, tak se se svým týmem určitě nezapomeňte přihlásit. Jarní soustředění se blíží a my už se moc těšíme na viděnou. :-)

Zdraví

Vaši milí organizátoři

Zadání úloh

Termín odeslání 5. série: 4. 4. 2017

Úloha 5.1 – Sčítací posloupnost (5b)

Postupně tvoříme posloupnost následujícím způsobem:

Začínáme se dvěma jedničkami. Mezi ně přidáme dvojku. Poté přidáme číslo 3 mezi všechna čísla, jejichž součet je 3. To samé opakujeme s dalšími čísly. Číslo n přidáme mezi všechna sousedící čísla v posloupnosti, jejichž součet je roven n. Po přidání čísla 5 tedy bude posloupnost vypadat následovně:

$$1, 5, 4, 3, 5, 2, 5, 3, 4, 5, 1.$$

Zjistěte, jak se dá charakterizovat počet výskytů čísla n (poté, co bylo do posloupnosti přidáno).

Úloha 5.2 – Pseudoatom (3b)

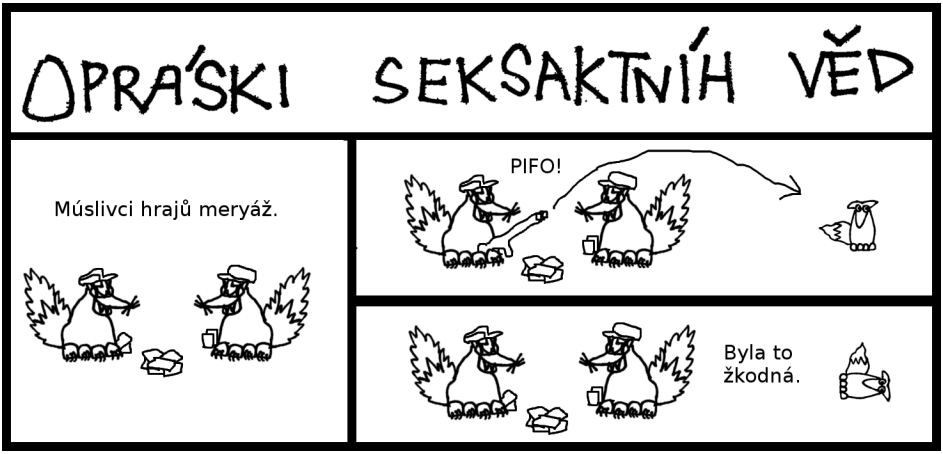
Mějme v blízkosti elektron a pozitron. Ty na krátký čas (přibližně 0,1 ns) vytvoří pseudoatom. Obě částice mají shodnou hmotnost m a opačný náboj e, resp. $-e$. Pohybují se po kruhové trajektorii okolo společného hmotného středu o poloměru r. Určete tento poloměr a vazebnou energii, t.j. energii nutnou pro oddálení částic do nekonečna. Řešte teoreticky pomocí Bohrova modelu atomu a číselně pro základní kvantový stav (tedy stav s nejnižší energií). Proč tento „atom“ není běžný při pozemských podmínkách?

Úloha 5.3 – Buchty (3b)

Komárodlak má obdélníkový pekáč s $m \times n$ buchtami, každá je buď maková, nebo tvarohová. Komárodlak si všimnul, že v každém řádku i v každém sloupci je lichý počet makových buchet.

1. Dokažte, že buď jsou m i n obě sudá, nebo obě lichá.

2. Kolik existuje pro pekáč o daných rozměrech rozmístění buchet takových, že splňují podmínku v zadání?



Úloha 5.4 – Rozdělení balíčku (3b)

Mám balíček n karet, každá karta je buď červená, nebo černá. Počty červených a černých karet nemusí být stejné. Balíček zamíchám a rozložím do řady. Chci určit, zda a kde můžu řadu rozdělit tak, aby vlevo od rozdělení bylo právě tolik červených karet, kolik je černých karet vpravo od rozdělení. Rychleji proveditelná řešení budou hodnocena lépe.

Bonusový úkol: Nevím, kolik karet mám, a místo rozložení do řady se na ně budu postupně koukat. Kolik nejméně bitů informace si potřebuji pamatovat v momentě, kdy jsem viděl už N karet, abych uměl reagovat jak na $(N+1)$-ní kartu, tak na informaci, že to již byly všechny karty? V prvním případě si chci nějak spočítat nové informace k zapamatování (staré pak zapomenu), v druhém chci odpovědět počtem karet, které je potřeba ze začátku balíčku odebrat, aby v odebrané části bylo tolik červených karet, kolik bude černých ve zbytku.

Řešení úloh 3. série

Úloha 3.1 – Odklízení cest (5b)

Zadání:

V džungli je mnoho rozcestí a pěšinek mezi nimi. Bohužel jsou cestičky zarostlé a Jonášovi s Jájou bude nějakou dobu trvat je opět zprůchodnit. Jája má mapku, která obsahuje informace o tom, jak dlouho jim bude trvat jednotlivé cestičky prosekat. Některá rozcestí jsou navíc označená, neboť by skrz ně mohla vést cesta ven z džungle. Na jednom takovém rozcestí aktuálně Jája s Jonášem stojí. Potřebovali by zjistit, kolik času budou muset strávit čištěním cest, než bude možné se dostat na všechna označená rozcestí. Čas potřebný na opětovné proběhnutí již jednou zprůchodněné cesty můžeme zanedbat. Také můžete předpokládat, že všechny cesty jsou průchozí v obou směrech a jejich prosekání bude trvat stejně dlouho nezávisle na tom, ze které strany postupujete.

Pro tento problém zatím nikdo neumí najít optimální řešení výrazně rychleji než zkoušením všech možností, kterých je opravdu hodně. Zkus ale najít algoritmus, který se k optimálnímu řešení přiblíží i v rozumném čase – třeba takový, jehož řešení bude v nejhorším případě dvojnásobkem optima a stihne ho nalézt v čase $\mathcal{O}(n^4)$, kde n je počet všech rozcestí na mapě.[1]

Pokud ti problém pořád přijde moc složitý, část bodů dostaneš i za řešení, které předpokládá, že Jonáš s Jájou potřebují navštívit všechna rozcestí.

Řešení:

Nejprve lehčí varianta: Pokud znáte problém hledání minimální kostry, tak se dalo nahlédnout, že se jedná právě o něj. K úspěšnému vyřešení to ale nebylo zapotřebí.

Vybereme si jedno rozcestí (například to, kde Jonáš s Jájou právě stojí), které prohlásíme za navštívené (všechna ostatní jsou nenavštívená) a zapamatujeme si všechny cesty, které z něj vedou. Dále budeme postupovat následujícím způsobem: Pokud už nemáme zapamatovanou žádnou neprosekanou cestu, tak to znamená, že žádné navštívené rozcestí nemá nenavštívené sousední rozcestí a tudíž jsme prohledali vše, co jsme mohli, a skončíme. Jinak ze všech ještě neprosekaných cest, které si pamatujeme, vybereme tu, kterou umíme prosekat nejrychleji. Pokud tato cesta vede mezi dvěma již navštívenými rozcestími, tak ji zapomeneme a celý postup opakujeme. V opačném případě spojuje navštívené rozcestí s nenavštíveným. Cestu prosekáme, její nenavštívený konec prohlásíme za navštívený a zapamatujeme si všechny ostatní cesty, které z něj vedou.

[1] To jest, algoritmus smí provést řádově nejvýš n^4 (elementárních) kroků. Může provést např. $42n^4$ kroků nebo $5n^4+3n^2+\log n$ kroků nebo n^2 kroků, ale už ne například n^5 kroků. Podrobněji se o měření efektivity algoritmů můžete dočíst v jedné z programátorských kuchařek: `http://ksp.mff.cuni.cz/kucharky/slozitost/`

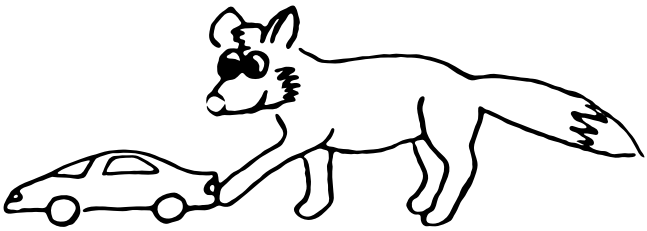
Důkaz správnosti: Pro spor nechť existují nějaké cesty, které jsme prosekali, ačkoliv nebyly součástí ideálního řešení. Z těchto cest vybereme tu, kterou jsme prosekali jako první. V momentě, kdy jsme tuto cestu prosekávali, se jednalo o nejsnadněji prosekatelnou cestu vedoucí z libovolného navštíveného rozcestí do libovolného nenavštíveného rozcestí.

Ideální řešení musí obsahovat posloupnost prosekaných cest, která vede z jednoho konce této „zbytečně“ prosekané cesty na druhý, a tato posloupnost nutně obsahuje cestu, o které jsme v době prosekání té zbytečně prosekané cesty věděli, ale tehdy jsme ji neprosekali. Existuje pouze jeden důvod, proč jsme něco takového mohli udělat – tu námi „zbytečně“ prosekanou cestu nebylo těžší prosekat než tu, kterou jsme tehdy neprosekali. Je dobré si uvědomit, že obě cesty mohly být stejně namáhavé k prosekání.

Můžeme tedy vzít ideální řešení, odstranit z něj tehdy neprosekanou cestu a přidat do něj námi „zbytečně“ prosekanou cestu. Tím se řešení nezhorší, zůstane platným a (což je důležité) odsuneme tím moment, kdy jsme se od ideálního řešení odchýlili. Tímto způsobem pak můžeme postupně odsouvat naše odchýlení od ideálního řešení tak dlouho, že k němu nikdy nedojde.

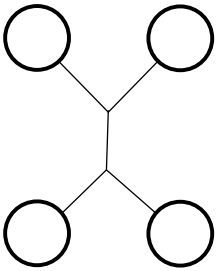
Časová náročnost bude záviset na tom, jak přesně si budeme pamatovat cesty, které možná ještě budeme chtít prosekat. Pokud použijeme obyčejnou haldu, dostaneme řešení v čase $\mathcal{O}(m \log m)$, kde m je počet cest v džungli. Tento výsledek se ještě dá několika triky vylepšit, zájemce si dovolím odkázat na skripta Krajinou grafových algoritmů[2].

Těžší varianta: Jedná se o problém Steinerova stromu. Použijeme Floydův-Warshallův algoritmus[3] k spočítání nejkratších cest mezi každými dvěma označenými vrcholy. Tento algoritmus běží v $\mathcal{O}(n^3)$. V úplném grafu tvořeném označenými vrcholy a váhami nejkratších cest mezi nimi nalezneme minimální kostru podobně, jako v lehčí variantě, a to v čase $\mathcal{O}(k^2 \log k)$, kde k je počet označených vrcholů, protože tento (úplný) graf obsahuje právě $k(k-1)/2 \leq k^2$ hran. Pro každou hranu této minimální kostry pak prosekáme odpovídající nejkratší cestu v původním grafu (kterou si můžeme pamatovat z Floyd-Warshallova algoritmu). Ježto $k^2 \log k$ je menší než n^3, celý algoritmus poběží ve stejném čase jako hledání všech nejkratších cest, tedy v $\mathcal{O}(n^3)$.

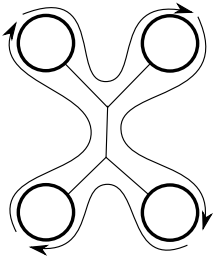


[2] `http://mj.ucw.cz/vyuka/ga/`

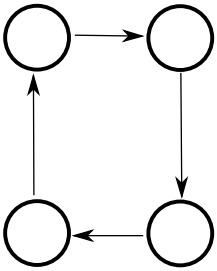
[3] Viz například `http://mj.ucw.cz/vyuka/ads/26-cesty.pdf`



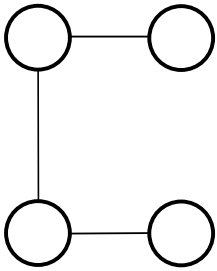
(a) Optimální řešení „velikosti“ OPT.



(b) Sled délky 2 OPT navštěvující všechny označené vrcholy.



(c) Sled navštěvující označené vrcholy ve stejném pořadí jako sled z předchozího obrázku, ale jdoucí nejkratšími cestami. Jeho délka je tedy nejvýše 2 OPT.



(d) Odstraněním některých hran z předchozího sledu dostaneme kostru v grafu označených vrcholů. Algoritmem nalezená minimální kostra nemůže být tedy na prosekání těžší.

Obrázek 1: Náznak důkazu, že námi nalezené řešení je nejhůře dvakrát tak těžké na prosekání, než optimální řešení.

Takhle nalezené řešení je nejvýše dvojnásobkem optima (viz Obrázek 1): Optimem je strom. Pokud uvážíme nejkratší sled[4] po optimem vybraných hranách, který navštěvuje všechny označené vrcholy a začíná a končí ve stejném vrcholu, tak zjevně každou hranu používá právě dvakrát. Pokud označené vrcholy budeme navštěvovat ve stejném pořadí, ale pokaždé půjdeme nejkratší cestou, tak jsme zjevně celkovou délku tahu neprodloužili a v nejhorším případě jsme každou hranu použili pouze jednou, což znamená, že jsme prosekáváním museli strávit nejvýše dvakrát tolik času. Naším algoritmem nalezené řešení, které z nejkratších cest mezi označenými vrcholy vybírá minimální kostru, pak určitě nebude horší.

Matej

[4]Sled v grafu je posloupnost vrcholů taková, že mezi každými po sobě jdoucími vrcholy vede hrana. Jak vrcholy, tak hrany se mohou ve sledu opakovat.

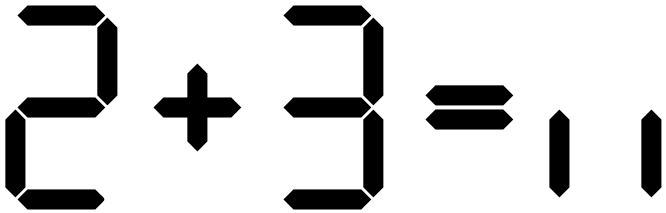
Úloha 3.2 – Displej (4b)

Zadání:
Máme vysílačku se sedmisegmentovým displejem, který dokáže zobrazit jedinou číslici, a s tlačítky 0, 1, . . . , 9. Přijdeme ke zhasnuté vysílačce a můžeme libovolně mačkat tlačítka. Každé zmáčknutí vždy změní všechny segmenty příslušné číslice na opačný stav: zhasnutý segment se rozsvítí a ten, co svítil, naopak zhasne. Po prvním zmáčknutí bude tedy na displeji svítit číslice z tlačítka. Pokud bychom jej zmáčkli znovu, displej se vrátí do původního, zhasnutého stavu. Další příklad je na Obrázku 2.

a) Vysílačka je bohužel už stará a jedno tlačítko se jí rozbilo. Dokážeš zobrazit na displeji číslici z rozbitého tlačítka pomocí těch ostatních? Je nějaké tlačítko nepostradatelné?

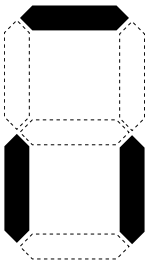
b) Ukaž, že je možné za použití všech tlačítek na displeji zobrazit libovolný obrazec, tedy rozsvítit libovolnou podmnožinu segmentů displeje. S jakou nejmenší skupinou funkčních tlačítek to je možné?

Za každou část úlohy můžeš získat dva body.



Obrázek 2: Po zmáčknutí tlačítek s číslicemi 2 a 3 bude na displeji zobrazený útvar v pravé části obrázku.

Řešení:
Nejdříve k části a). Zaměříme se na skupiny číslic takové, že když stiskneme odpovídající tlačítka, tak nakonec na displeji zhasnou všechny segmenty. Takovými skupinami jsou například 02468, 12369 a 5689. Pokud se rozbije nějaké tlačítko, tak stačí najít skupinu, ve které se tlačítko nachází, a zmáčknout všechna ostatní tlačítka z této skupiny. Protože nezáleží na pořadí stisknutí tlačítek, lze snadno nahlédnout, že nakonec na displeji bude svítit chybějící číslice. Všimněme si, že naše nalezené skupiny obsahují všechny číslice kromě 7, umíme tedy jakoukoliv zvolenou číslici kromě 7 nahradit ostatními. Sedmička je nepostradatelná, protože se jako jediná liší horním segmentem od dolního segmentu. Všechna ostatní tlačítka buď rozsvítí/zhasnou horní i dolní segment zároveň, nebo ho neovlivní vůbec, takže nemohou vytvořit číslici 7.



Obrázek 3: Ukázkový obrazec na displeji, který zkonstruujeme v úloze 3.2.

Nyní k části b). Základní postřehy: Pokud libovolné tlačítko zmáčkneme dvakrát, je to stejné, jako kdybychom ho nezmáčkli vůbec. Nezáleží na pořadí stisknutí tlačítek. Takže pokud bychom například postupně namačkali 1, 2, 7, 8, 9, 2, 1, 8, 8, 8, vede to ke stejnému výsledku jako zmáčknutí pouze 7 a 9. Z toho plyne, že má smysl pouze řešit, která podmnožina tlačítek dá jaký výsledek na displeji.

A teď už se budeme věnovat otázce, zda lze na displeji zobrazit libovolný obrazec. Pro ilustraci si řekněme, že chceme rozsvítit obrazec z Obrázku 3.

Sestavíme si rovnice pro stav každého segmentu displeje. Jednotlivé neznámé x_0 až x_9 nabývají hodnoty 0, pokud příslušná číslice nebyla zmáčknuta, a hodnoty 1, pokud příslušná číslice byla zmáčknutá. Budeme mít tedy soustavu sedmi rovnic o deseti neznámých.

Všechny naše výpočty budeme provádět nad dvouprvkovým tělesem $\mathbb{Z}_2$, kde sčítání funguje jednoduše: $0{+}0 = 0$, $0{+}1 = 1$, $1{+}0 = 1$, $1{+}1 = 0$, což je známo také jako operace XOR (čísla 0 a 1 budeme používat jako koeficienty před proměnnými, byť je nebudeme explicitně psát, takže mějme na paměti, že například $x_3{+}x_3 = 0$). Níže uvedené rovnice vlastně vyjadřují, že pokud některý segment má svítit (1), tak z množiny číslic, které ho ovlivňují, musíme zmáčknout lichý počet. Naopak pokud segment nemá svítit (0), tak z jemu odpovídající množiny číslic musíme zmáčknout sudý počet. Podívejme se na naši soustavu rovnic. Pro horní vodorovný segment máme:

$$x_0 + x_2 + x_3 + x_5 + x_6 + x_7 + x_8 + x_9 = 1,$$

pro levý horní segment:

$$x_0 + x_4 + x_5 + x_6 + x_8 + x_9 = 0,$$

pro pravý horní segment:

$$x_0 + x_1 + x_2 + x_3 + x_4 + x_7 + x_8 + x_9 = 0,$$

pro prostřední vodorovný segment:

$$x_2 + x_3 + x_4 + x_5 + x_6 + x_8 + x_9 = 0,$$

pro levý dolní segment:

$$x_0 + x_2 + x_6 + x_8 = 1,$$

pro pravý dolní segment:

$$x_0 + x_1 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 = 1,$$

a nakonec pro dolní vodorovný segment:

$$x_0 + x_2 + x_3 + x_5 + x_6 + x_8 + x_9 = 0.$$

Budeme aplikovat sčítací metodu. Začneme tím, že první rovnici přičteme ke všem rovnicím obsahujícím x_0 a obdržíme soustavu šesti rovnic o devíti neznámých:

$$\begin{aligned}
x_2 + x_3 + x_4 + x_7 &= 1\\
x_1 + x_4 + x_5 + x_6 &= 1\\
x_2 + x_3 + x_4 + x_5 + x_6 + x_8 + x_9 &= 0\\
x_3 + x_5 + x_7 + x_9 = &= 0\\
x_1 + x_2 + x_4 &= 0\\
x_7 &= 1
\end{aligned}$$

Nyní se zbavíme x_1 tím, že druhou rovnici sečteme s předposlední (čímž nám předposlední rovnice zmizí):

$$\begin{aligned} x_2 + x_3 + x_4 + x_7 &= 1 \\ x_2 + x_5 + x_6 &= 1 \\ x_2 + x_3 + x_4 + x_5 + x_6 + x_8 + x_9 &= 0 \\ x_3 + x_5 + x_7 + x_9 &= 0 \\ x_7 &= 1 \end{aligned}$$

Necháme zmizet x_7 dosazením jedničky za x_7 do všech rovnic, které x_7 obsahují (což není nic jiného než přičtení poslední rovnice):

$$\begin{aligned} x_2 + x_3 + x_4 &= 0 \\ x_2 + x_5 + x_6 &= 1 \\ x_2 + x_3 + x_4 + x_5 + x_6 + x_8 + x_9 &= 0 \\ x_3 + x_5 + x_9 &= 1 \end{aligned}$$

Teď přičteme první rovnici ke druhé a třetí (čímž se první rovnice zbavíme), takže odstraněním x_2 dostaneme soustavu 3 rovnic o 6 neznámých:

$$\begin{aligned} x_3 + x_4 + x_5 + x_6 &= 1 \\ x_5 + x_6 + x_8 + x_9 &= 0 \\ x_3 + x_5 + x_9 &= 1 \end{aligned}$$

Sečteme první s třetí rovnicí, čímž se nadobro zbavíme x_3:

$$\begin{aligned} x_4 + x_6 + x_9 &= 0 \\ x_5 + x_6 + x_8 + x_9 &= 0 \end{aligned}$$

Provedeme poslední sečtení, abychom se zbavili x_6:

$$x_4 + x_5 + x_8 = 0$$

Při posledním kroku nám krom x_6 vypadlo i x_9, takže se můžeme rozhodnout, že bude vždycky nulové. V poslední rovnici si zase můžeme říct, že pouze hodnotu x_4 přizpůsobíme pravé straně a proměnné x_5 a x_8 budou vždy nulové. Získáme tedy:

$$\begin{aligned} x_4 &= 0 \\ x_4 + x_6 &= 0 \\ x_3 + x_4 + x_6 &= 1 \\ x_2 + x_3 + x_4 &= 0 \\ x_2 + x_3 + x_4 + x_7 &= 1 \\ x_1 + x_4 + x_6 &= 1 \\ x_0 + x_2 + x_3 + x_6 + x_7 &= 1 \end{aligned}$$

Zpětná substituce nám postupně dá:

$$x_4 = 0, \quad x_6 = 0, \quad x_3 = 1, \quad x_2 = 1, \quad x_7 = 1, \quad x_1 = 1, \quad x_0 = 0.$$

A opravdu! Z jedničky, dvojky, trojky a sedmičky vznikne náš požadovaný symbol. Nyní si všimněme, že na pravých stranách rovnic vlastně vůbec nezáleželo, protože jsme je jen mechanicky přičítali a projevily se až nakonec ve zpětné substituci. Kdybychom si zvolili jiné pravé strany, tak by nám jen vyšla odlišná čísla, ale postup našich úprav by byl stejný. Zejména tedy stále platí, že za x_9, x_5 a x_8 můžeme pořád dosazovat nulu. Znamená to tedy, že tlačítka pro pětku, osmičku a devítku nikdy nepotřebujeme použít a naše rovnice popíší, jak libovolný zadaný znak postavit z číslic 0, 1, 2, 3, 4, 6, 7.

Zjistili jsme tedy, že libovolný obrazec dokážeme sestavit s využitím sedmi tlačítek. S menším počtem tlačítek to už není možné, protože existuje 2^7 různých obrazců, ale 6 tlačítek by nám dalo pouze 2^6 kombinací, které lze namačkat.

Martin

Úloha 3.3 – Vážení (3b)

Zadání:
Popište, co bude ukazovat displej digitální tenzometrické váhy, když se na ni postaví mužík s těžkým kulovým závažím v rukou a zdvihne jej pomalu nad hlavu. Co budeme pozorovat, když závaží upustí a během pádu znovu chytí u kolen?

Chybějící parametry a jejich vliv na průběh děje odhadněte na základě skutečného světa, případně si pomozte experimentem. Můžete také popsat možný rozsah očekávaných výsledků pozorování. Nezapomeňte vaše úvahy náležitě zdůvodnit a závěry vysvětlit.

Řešení:
Tenzometrické váhy vnímajú silu, ktorá je na nich vyvíjaná. Preto hmotnosť M, ktorú uvidíme je daná vzťahom:

$$M = \frac{F}{g} = \frac{mg}{g},$$

kde g je tiažové zrýchlenie a m hmotnosť objektu na váhe. Zo zákonu akcie a reakcie dostaneme, že ak niečo urýchlime proti smeru g, narastie hmotnosť M na váhe. Z toho dostaneme vzťah, z ktorého budeme vychádzať:

$$M = \frac{m_{\check{c}}(g - a_{\check{c}})}{g} + \frac{m_z(g - a_z)}{g},$$

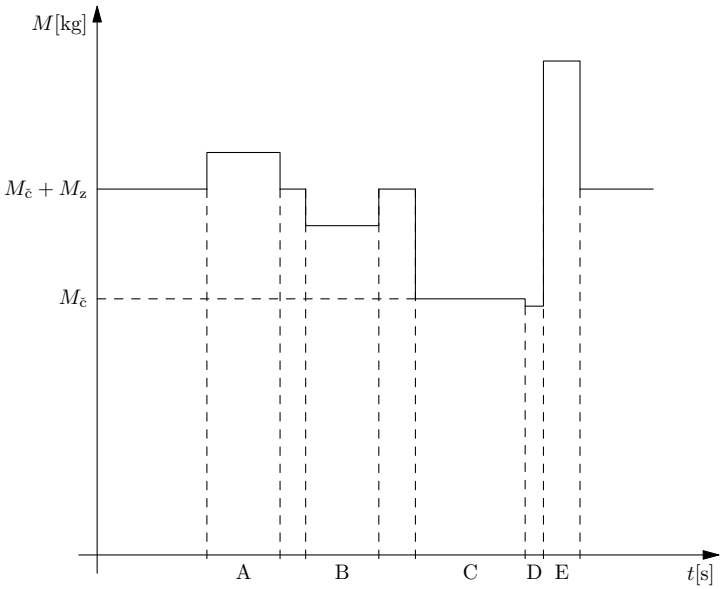
kde $m_{\check{c}}$ a $a_{\check{c}}$ je hmotnosť človeka a jeho zrýchlenie a m_z a a_z hmotnosť a zrýchlenie závažia.

Z tohto vzťahu jednoducho odvodíme, čo sa bude diať, keď zdvihneme závažie nad hlavu. Pri začiatku zdvíhania urýchlime závažie a váha ukáže zvýšenú hmotnosť M počas urýchľovania. Následne nastane rovnomerný pohyb, kde zrýchlenie bude nulové tak uvidíme počiatočnú hmotnosť $M = m_{\check{c}} + m_z$. Nakoniec, keď budeme zastavovať závažie nad hlavou, hmotnosť M klesne počas zastavovania. Na zmene M bude mať podiel aj človek, keďže bude urýchľovať a spomaľovať minimálne svoje ruky.

Pri pustení závažia váha ukáže iba hmotnosť $m_{\check{c}}$. Človek, aby chytil niečo u kolien sa musí pripraviť, preto urýchli a následne spomalí niektoré časti svojho teľa. Z toho môžeme vidieť kolísanie ukázanej hmotnosti na váhe do vyššej aj nižšej hodnoty od $m_{\check{c}}$.

Pri chytení a brzdení závažia uvidíme zväčšenie hmotnosti podľa veľkosti brzdenia. Čím bude spomalenie väčšie uvidíme vyšší a užší vrchol v grafe $M(t)$. Hodnota M sa nakoniec ustáli na $m_{\check{c}} + m_z$. Ako v minulých odstavcoch bude človek prispievať k tomuto javu.

Pre overenie teórie sa postavím na váhu a vidím približne 85 kg, do rúk si zoberiem závažie s hmotnosťou približne 5 kg a vidím výslednú hmotnosť 90 kg. Viackrát zopakujem pohyb zavažím od hrude nad hlavu kolmo nahor. Vidím výchylky v rozsahu pár kilogramoch podľa prudkosti pohybu. Najprv do väčších



Obrázek 4: Znázornení graf ukazuje predpokladanú závislosť $M(t)$. Tento graf je iba ilustračný, kde výkyvy od stálej hmotnosti budú mať tvar podľa priebehu zrychlení človeka a závažia, ktoré na nich pôsobia, zavislích od času.

A – Zrýchlenie závažia pri pohybe nad hlavu

B – Zastavenie závažia nad hlavou

C – Upustenie závažia

D – Príprava človeka na chytenie závažia pri kolenách

E – Zastavovanie závažia

hodnôt pri začiatku pohybu a potom do nižších hodnôt a následne ustálenie na 90 kg. Výchylky do vyšších hodnôt sú výraznejšie, podľa mna spôsobené aj mojou nedokonalosťou a prispievaním zrýchlením častí môjho tela. Iba pri pomalom pohybe závažia sa mi podarilo vidieť 90 kg na váhe medzi zrychlením a spomalením. Následne závažie dám trochu predomňa ale stále vo výške hlavy a pustím ho. Vidím takmer okamžite hmotnosť 85 kg a pripravujem sa na chytenie pri kolenách. Na váhe vidím pokles aj o niekoľko kilogramov. Musím sa zohnúť a dostať ruky z polohy nad hlavou ku kolenám pred zavažím. Tesne pred chytením už vidím narastať hmotnosť ako sa pripravujem a brzdím svoje telo a ruky v pohybe dole. Po chytení vidím hodnoty až nad 110 kg pri rýchlom chytení. Pri pomalom chytení vidím hmotnosť okolo 100 kg, ale spolu so zavažím sa pohybujem smerom dolu až do jeho zastavenia. Pri každom pokuse sa mi určite nepodarilo udržať konštantné zrýchlenie alebo spomalenie závažia. Experiment potvrdil naše teoretické predpovede.

Kubo

Úloha 3.4 – Sumy a čtverce (5b)

Zadání:

a) Součet čtyř reálných čísel je 8 a součet jejich druhých mocnin je 16. Jaká je největší možná hodnota největšího čísla? Tedy pro reálná a, b, c, d platí: $a + b + c + d = 8$ a $a^2 + b^2 + c^2 + d^2 = 16$. Najdi největší možnou hodnotu čísla a.

b) Součet pěti reálných čísel je 8 a součet jejich druhých mocnin je 16. Jaká je tentokrát největší možná hodnota největšího čísla?

Řešení:

Jak už to v úlohách tohoto typu bývá, (jednodušší) půlkou řešení je uhodnout správný výsledek. Konkrétně v první podúloze není těžké najít jednu vyhovující čtveřici $a = b = c = d = 2$. Dokonce platí, že je to jediné řešení této soustavy rovnic. V druhé podúloze si můžeme tipnout, že v případě, kdy maximalizujeme proměnnou a, budou všechny ostatní proměnné nabývat stejné hodnoty, tedy $b = c = d = e$. Za tohoto předpokladu snadno dopočítáme dvě možná řešení rovnic – první z nich, $a = 0$, $b = c = d = e = 2$, odpovídá předchozí podúloze. Druhé z nich, $a = 3{,}2$, $b = c = d = e = 1{,}2$, je skutečně hledaným extrémním případem. Teď už jen zbývá předchozí tvrzení dokázat.

Trikové vzorové řešení

a) Uvažme kvadratický výraz $(a-2)^2 + (b-2)^2 + (c-2)^2 + (d-2)^2$, který můžeme upravovat s pomocí rovnic ze zadání:

$$\begin{aligned}(a-2)^2 + (b-2)^2 + (c-2)^2 + (d-2)^2 = \\ a^2 + b^2 + c^2 + d^2 - 4 \cdot (a+b+c+d) + 4 \cdot 2^2 = \\ 16 - 32 + 16 = 0\end{aligned}$$

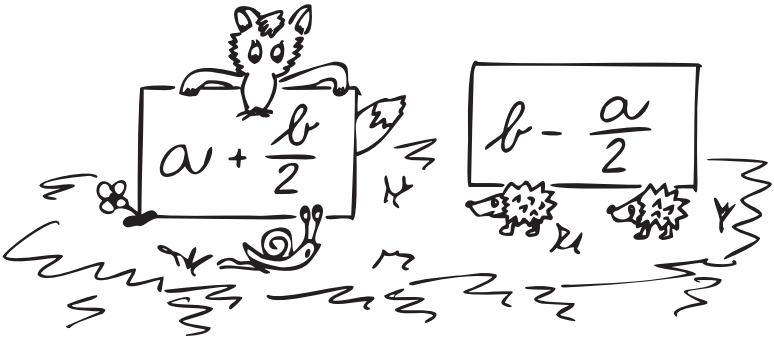
Výraz je součtem čtverců, takže aby byl rovný nule, musí být rovny nule všechny výrazy uvnitř závorek. Tedy pro naše proměnné musí platit $a = b = c = d = 2$. Tato čtveřice rovnicím vskutku vyhovuje.

b) Uvažme kvadratický výraz $(a-1{,}2)^2 + (b-1{,}2)^2 + (c-1{,}2)^2 + (d-1{,}2)^2 + (e-1{,}2)^2$, který můžeme upravovat s pomocí rovnic ze zadání:

$$\begin{aligned}(a-1{,}2)^2 + (b-1{,}2)^2 + (c-1{,}2)^2 + (d-1{,}2)^2 + (e-1{,}2)^2 = \\ a^2 + b^2 + c^2 + d^2 + e^2 - 2{,}4 \cdot (a+b+c+d+e) + 5 \cdot 1{,}2^2 = \\ 16 - 19{,}2 + 7{,}2 = 4\end{aligned}$$

Tedy musí platit $(a-1{,}2)^2 \leq 4$, z toho plyne $a \leq 3{,}2$, a všechny proměnné tak musí být menší či rovny číslu 3,2. Na druhou stranu řešení $a = 3{,}2$, $b = c = d = e = 1{,}2$ obě rovnice vskutku splňuje.

Jak na něco takového přijít? Rovnice ze zadání nám umožňují vyhodnotit jednoduché symetrické kvadratické výrazy. V obou podúlohách jsme tedy zkonstruovali výraz, jehož hodnotu známe. Zároveň jsme ale členy tohoto výrazu zvolili tak, aby se jejich hodnota dala snadno odhadnout (druhé mocniny jsou nezáporné). Navíc v našem odhadu dojde k rovnosti, pokud za proměnné dosadíme naše uhodnuté řešení (o kterém doufáme, že je hledaným extrémním řešením). Když jsme tedy v druhé podúloze výrazy $(b-1{,}2)^2, \ldots, (e-1{,}2)^2$ odhadli nulou, pro naše uhodnuté řešení tyto výrazy byly rovny nule. Proto i výsledný odhad na proměnnou a odpovídal našemu uhodnutému řešení.



Řešení využívající Cauchy-Schwarzovu nerovnost (podle Dr.$^{\mathsf{MM}}$Tomáše Domese a Mgr.$^{\mathsf{MM}}$Filipa Čermáka)

K řešení šla také použít Cauchy-Schwarzova (CS) nerovnost použitá na všechny proměnné ze zadání kromě a (opět zde využíváme myšlenky, že v extrémním případě budou tyto proměnné nabývat stejné hodnoty a v CS nerovnosti tak dojde k rovnosti). Ukážeme pouze řešení druhé podúlohy (ta první by se vyřešila analogicky). První podúloha šla také řešit přímým použitím CS nerovnosti pro všechny proměnné.

Z Cauchy-Schwarzovy nerovnosti plyne:

$$(b^2+c^2+d^2+e^2)\cdot\left(\frac{1}{4}+\frac{1}{4}+\frac{1}{4}+\frac{1}{4}\right)\geq\left(\frac{b}{2}+\frac{c}{2}+\frac{d}{2}+\frac{e}{2}\right)^2$$

Po úpravě a dosazení $b^2+c^2+d^2+e^2=16-a^2$ a $b+c+d+e=8-a$ dostáváme

$$16-a^2\geq\left(\frac{8-a}{2}\right)^2,$$

tedy

$$5a^2-16a\leq 0.$$

Z této nerovnice již vyplývá $a\leq 3{,}2$, zbytek je stejný jako ve vzorovém řešení.

Vašek

Řešení témat

Téma 3 – Filmoví poradci

K tématu přišel příspěvek od Mgr.MM Terezy Hladíkové. Rozebírala v něm chyby v seriálu a filmech Star Trek. Zabývala se warpovým pohonem, transportérem a únikem zpod horizontu událostí černé díry.

Zatím se při řešení tématu nikdo nezabýval cestováním v čase ve filmech. Když Marty McFly ve filmu Návrat do budoucnosti cestuje do minulosti, změní ji, a kvůli tomu začne mizet. Oproti tomu ve filmu Harry Potter a vězeň z Azkabanu se při cestování do minulosti odehraje to, co se už vlastně stalo, takže v ní nemají protagonisté svobodnou vůli. Zamyslete se, jaké jsou v těchto nebo ostatních filmech fyzikální a logické nesmysly a sepište, jak by nejspíš vypadala cesta do minulosti, kdyby byla možná. (Většina fyzikálních teorií počítá s tím, že do minulosti cestovat nelze, avšak pokud cesty časem povolíme, nemusíme dostat spor a můžou nastat jevy, které bez stroje času neznáme.) Můžeme cestovat do padesátých let a zastřelit svého dědečka? Nezapomeňte popsat, jak jste svoje hypotézy vymysleli, případně uvést zdroje.

Viktor

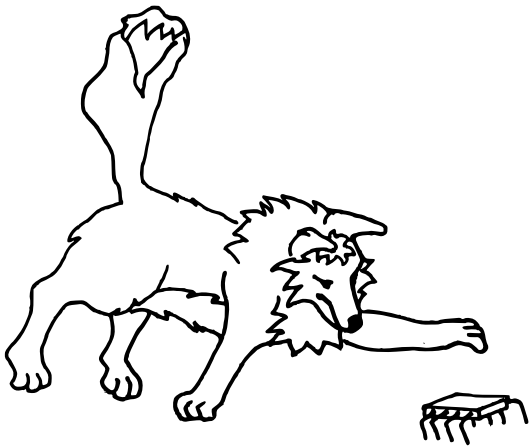
Seriál: Malý úvod do assembleru

Nejprve se na chvíli vrátím k instrukční sadě, kterou jste navrhovali. Konkrétně k jednomu z možných řešení malé délky instrukcí, které se dnes používá. Jsou jím únikové sekvence. Myšlenka je prostá – procesor přečte instrukci a podívá se například na její první bit. Pokud je tento bit nulový, procesor instrukci okamžitě vykoná. Pokud ale nalezne hodnotu jedna, uloží instrukci do předem určeného registru a do jiného registru (například do registru `F`) si poznamená, že narazil na únikovou sekvenci a dále již nic nevykonává. V příštím kroku pak nebere v potaz jen přečtenou instrukci, ale i poznamenanou hodnotu. Konkrétní příklad pro `ADD Rd, Rr` z minula:

```
ESC A, B  ; zahajuje únikovou sekvenci, do registru E
          ; jsou zapsány adresy A, B
          ; do registru F příznak únikové sekvence
ADD       ; najednou máme k dispozici celých 8 bitů
          ; na zakódování instrukce -- a tedy
          ; pro nás najednou není problém zakódovat
          ; libovolných 256 instrukcí se
          ; dvěma operandy
```

Únikové sekvence se dají zobecnit a vedou k instrukcím o různých délkách – ty, které se používají často, budou krátké, zatímco obskurnosti budou delší řetězce. Tato vlastnost je jak jejich největší výhodou, tak jejich strašlivou slabinou. Výhodou je možnost mít mnoho rozličných instrukcí s délkou závislou na užitečnosti a specifičnosti instrukcí. Nevýhodou je značná nepřehlednost výsledného strojového kódu a menší kontrola nad jeho strukturou.

Za zmínku stojí rozštěpení architektur procesorů na dva druhy: RISC a CISC. První skupina, jejíž název je zkratkou z Reduced Instrucion Set Computer, se rozhodla, že do procesoru zabuduje instrukcí jen málo. Tento přístup má výhodu v přehlednosti a jednoduchosti. Na druhou stranu si toho hodně musí zařizovat programátor sám. Druhá skupina, Complex Instruction Set Computers, usoudila, že co zadrátovat půjde, to zadrátují. Takže jejich procesory podporují velmi široké spektrum instrukcí – třeba maticové násobení nebo modifikace většího rozsahu adres paměti naráz.



Ač jsou CISCové procesory nepochybně zajímavé, pro naše účely jsou zbytečně komplikované. Náš *PAPIRAS* je tedy assembler RISCový – instrukcí bude málo. Jak jsme si minule ukázali, 8 bitů nám nebude stačit. Od této chvíle přecházíme na 16 bitů (tedy registry, paměť i instrukce nyní budou mít 16 bitů). To pořád není nic extra, ale už je to na jedné straně dostatečné pro většinu praktických účelů a na straně druhé dostatečně omezující na to, abychom nezpohodlněli – jeden z důvodů naší snahy naučit se práci s assemblerem je práce s omezenými zdroji. Ještě krátce odbočím do historie, abych vás dostatečně motivoval ke krokování na papíře. Původní assembler totiž sloužil jen jako mnemotechnická pomůcka pro lepší zapamatování instrukcí – programátor si sepsal kód v assembleru, potom uchopil do jedné ruky instrukční tabulku k procesoru a do druhé psadlo a jal se přepisovat assembler na jedničky a nuly. A nesměl udělat chybu. Oproti tomu je naše krokování docela jednoduché, ne?

Instrukce	Popis
`NOP`	No operation – procesor nedělá nic
`LDI R, k`	`R` ← `k` (zamyslete se, je konstanta `k` něčím omezena? Čím?)
`INC R`	Přičte k registru jedna. Operace přetéká.
`DEC R`	Odečte od registru jedna. Operace podtéká.
`ADD Rd, Rr`	`Rd` ← `Rd` + `Rr` operace přetéká
`SUB Rd, Rr`	`Rd` ← `Rd` − `Rr` operace podtéká
`ORI Rd, k`	`Rd` ← `Rd` OR `k`, bitový or
`ANDI Rd, k`	`Rd` ← `Rd` AND `k`, bitový and
`NOT Rd`	`Rd` ← NOT `Rd`, bitová negace
`OR Rd, Rr`	`Rd` ← `Rd` OR `Rr`, bitový or dvou registrů
`AND Rd, Rr`	`Rd` ← `Rd` AND `Rr`, bitový and dvou registrů
`MOV Rd, Rr`	`Rd` ← `Rr`
`SHL Rd, k`	`Rd` ← `Rd` $\ll$ `k`, bitový posun doleva
`SHR Rd, k`	`Rd` ← `Rd` $\gg$ `k`, bitový posun doprava
`STOP`	ukončí chod procesoru

Tabulka 1: Seznam instrukcí. Velkým písmenem značíme registr, malým konstantu.

Jdeme programovat. Podívejte se tedy prosím na Tabulku 1. Jedná se o trochu rozšířený seznam instrukcí z minula, pro připomenutí. Také se nám bude hodit připomenout si náš procesor, který ale oproti minulému dílu rozšíříme o jeden registr. Tento registr se bude nazývat `Flags` (Příznaky) a budeme jej označovat zkratkou `Fl`.

Můžeme se k němu chovat stejně jako k registrům `A` až `H`, ale spíše to dělat nebudeme – procesor jej totiž používá k ukládání informací – tzv. flagů. Pár malých programů jste si měli zapsat minule – a tehdy jste nejspíše zjistili, že *PAPIRAS* toho zatím neumí ani tolik jako obyčejná kalkulačka. Zejména programátorům také chyběla možnost kontrolovat chod programu – tedy větvení a skoky. Je tedy na čase si rozšířit instrukční sadu.

Budiž skoky. Skoky jsou instrukce, které modifikují registr `PC`. Rozlišujeme dva druhy – skoky absolutní a relativní.

`JMP A` – skok na adresu obsaženou v registru A je prototyp absolutního skoku. Nevýhodou je nutnost předem do nějakého registru uložit adresu, ale díky tomu můžeme skočit kamkoliv v rozsahu dostupné paměti.

`JMPA k` – k současné adrese přičte konstantu `k`

`JMPS k` – od současné adresy odečte konstantu `k`

Poslední dva skoky patří mezi skoky relativní – skáčeme o nějaký kus dopředu nebo dozadu. Dokud neřeknu jinak, budeme první instrukci nahrávat na adresu 0 a další hned za ni. Instrukce relativních skoků nemohou skákat příliš daleko –

jsme omezeni délkou instrukce, takže můžeme skákat třeba o 1024 adres[5]. Nebudu vás nutit to dodržovat, ale rád bych, abyste si to uvědomili. Skok nám umožní například generovat Fibbonacciho čísla v registrech `A, B`:

```
LDI B, 1     ;adresa 0x0
LDI C, 0x2
ADD B, A     ;adresa 0x2
ADD A, B
JMP C        ;nahraje do PC registru hodnotu 0x2 a
             ;dále tedy pokračujeme zase
             ;od instrukce ADD B, A
```

Zkuste si to odkrokovat a potom vymyslete, jak celý kód o instrukci zkrátit.

Se skoky úzce souvisí pojem návěstí (label). Jedná se o zkratku pro překladač, který si s její pomocí dopočítá posun, o který má skočit. Abychom nemuseli myslet na to, jestli skákat s pomocí `JMPA` nebo `JMPS` podle umístění návěstí, zavedeme si zobecňující instrukci `JMPR label` (Jump Relative), která skočí na návěstí nezávisle na jeho umístění. (Ve skutečnosti by si překladač vybral, kterou z instrukcí

[5] Záleží na instrukční sadě a procesoru. Pro většinu našich účelů to bude stačit a když ne, máme absolutní skok.

JMPA, JMPS ji má nahradit pro dané návěstí). Používání návěstí doporučuji, je to pohodlnější. Návěstí také můžeme nahrát do registru pomocí instrukce LDI – v tom případě budeme mít v registru uloženu přímo jeho adresu, využitelnou následně pro absolutní skoky. Při použití návěstí by předchozí příklad vypadal takto:

```
LDI B, 1
LDI C, vypocet      ;alternativně by bylo možno
                    ;tuto instrukci vynechat
vypocet:
ADD B, A            ;a použít místo absolutního
                    ;skoku skok relativní
ADD A, B
JMP vypocet
```

Ale skoky, jakkoliv užitečné, nám samy o sobě stačit nebudou – místo rovné cesty získáme cestu, která se bude všelijak kroutit, ale ještě stále neumíme odbočit. Nemůžeme se rozhodovat. K programování ale nějaký způsob, jak řídit chod programu, potřebujeme. A k tomu nám bude sloužit následující instrukce: BREQ label – Branch If Equal. Skoč, pokud se něco rovnalo. Ale co se mělo rovnat? A jak to zjistíme? Vzpomeňte si na registr Flags (Fl). Ten se nám teď bude hodit – procesor si do něj totiž ukládá některé informace o předchozích operacích. Pro jeden z jeho bitů se používá zkratka Z – ten je nastaven na hodnotu 1 v případě, že výsledek předchozí aritmetické operace byl nula; pokud tento výsledek nebyl nula, je bit naopak vynulován. Tedy BREQ skočí, pokud je bit Z registru Fl nastaven na hodnotu 1. Druhá instrukce ze stejného soudku je BRNE label, Branch If Not Equal, tedy opak BREQ.

Teď už toho umíme docela dost – nějaké základní aritmetické operace, skoky v programu a řízení toho, kam skáčeme. S tím už by se možná dalo udělat něco zajímavějšího.

Úloha 5.5 – Programování v Assembleru (5b)

Podúloha 5.5.1 – Tisíckrát nic (2b)

Napište kus kódu, který poběží přesně tisíc instrukcí a bude co nejkratší co do spotřebované paměti. Nejhorší možné řešení je tedy napsat 999 krát instrukci `NOP` následovanou instrukcí `STOP`.
Bonus: Napište kus kódu s parametrem (uloženým v registru `H`), který určí, kolik instrukcí se má vykonat, než skončí.

Podúloha 5.5.2 – Ach ta paměť (1b)

Registry jsou sice jednoduše přístupné, ale přeci jen je jich poněkud málo. Paměť k ukládání programu je sice poměrně velká, ale zatím do ní neumíme zapisovat. Co když budeme chtít ukládat větší množství dat? (Například při periodickém čtení nějakého senzoru.) Paměť pro data, stejně jako paměť pro program, je páska s adresami, kde každé adrese odpovídá jedno 16bitové políčko. Zamyslete se nad čtením a zápisem dat do paměti. Tedy navrhněte instrukci pro čtení a instrukci pro zápis (nebo více různých instrukcí).

Když jsme si řekli, že datová paměť vypadá stejně jako paměť programová, je vůbec nějaký důvod je mít oddělené? A pokud oddělené nebudou, jaké to bude mít důsledky?

Podúloha 5.5.3 – Malý velký Indián (2b)

S pomocí instrukcí pro práci s pamětí zkuste do paměti uložit 32bitové číslo. (Pozor, jedno políčko paměti má 16 bitů, co s tím?) Máte? Uložte ještě jedno. A teď je zkuste sečíst. Možná narazíte na problém s přenosem bitu z jedné poloviny výsledku do druhé. Pomůže nám opět registr `Flags` – instrukce `ADD` nastaví jeho carry bit, pokud při sčítání bezznaménkových čísel dojde k přetečení. Přístup k němu je sekvence instrukcí (načtu carry bit do registru `A`)

```
MOV A, Fl
ANDI A, 0b1 ;(carry bit je nultý bit registru Flags)
```

Pokud vám ani to nepomůže, použijte instrukci `ADC Rd, Rr` – Add With Carry, která provede operaci `Rd ← Rd + Rr + Carry bit registru Flags` (přesněji `Rd + Rr + (Fl AND 0b1)`).
Bonus: Použili jste při ukládání do paměti Little Endian nebo Big Endian? A co jsou to ti endiáni?

A to bude pro dnešek vše. Skončili jsme přesně na hranici, za kterou věci začínají být velmi zajímavé, ale za kterou už jsou končiny poměrně obtížně schůdné.

Tomáš Bartoněk

Výsledková listina 3. čísla

Poř.	Jméno	R.	$\sum_{-1}$	Úlohy r1	r2	r3	r4	t3	$\sum_0$	$\sum_1$
1.	Dr.MMA. Mlezivová	3	82,7		3,7	1,7	2,0		7,4	27,2
2.	Dr.MMP. Turinský	4	51,2	2,5	3,1				5,6	25,7
3.	Mgr.MMK. Rosická	2	25,5		3,5				3,5	25,5
4.	Mgr.MMK. Balej	2	25,1		3,4	2,9	2,0		8,3	25,1
5.	Mgr.MMK. Čížková	3	29,1		3,6		2,0		5,6	22,6
6.	Mgr.MMJ. Suchánek	3	35,1	4,5	3,7				8,2	19,9
7.–8.	Bc.MMM. Holubička	1	18,3		2,0				2,0	18,3
	Bc.MMV. Pavlíček	1	18,3						0	18,3
9.	Bc.MMB. Hroncová	2	18,2		1,2	2,0			3,2	18,2
10.–11.	Mgr.MMJ. Paidar	3	35,8		3,4		3,0		6,4	16,7
	Bc.MMK. Vnuková	2	16,7						0	16,7
12.	Bc.MMR. Olšák	2	15,8		3,8				3,8	15,8
13.	Doc.MMP. Šimůnek	4	157,0			1,5			1,5	14,3
14.	Mgr.MMF. Čermák	3	49,7				4,9		4,9	14,1
15.	Bc.MMZ. Lüköová	2	13,1						0	13,1
16.	Bc.MMM. Bukvaj	1	13,0						0	13,0
17.	Bc.MMJ. Růžička	2	18,9		3,5	2,9	1,0		7,4	12,9
18.	Mgr.MMT. Hladíková	3	48,4					3,0	3,0	12,8
19.	Bc.MMO. Gonzor	Z9	11,2						0	11,2
20.	Mgr.MMB. Požár	3	30,9						0	10,8
21.	Mgr.MML. Kopfová	2	29,7		2,0		2,0		4,0	10,7
22.	Bc.MMT. Kulichová	2	10,2						0	10,2
23.	Mgr.MMJ. Pallová	2	28,8						0	9,5
24.	Mgr.MML. Kundratová	2	36,7		4,0				4,0	9,4
25.	Mgr.MMO. Buček	3	27,6						0	9,3
26.–27.	Bc.MMA. Neubauerová	3	17,0						0	9,0
	V. Procházka	3	9,0						0	9,0
28.	T. Drobil	2	8,7						0	8,7
29.–30.	Dr.MMT. Domes	4	85,8		3,4		5,0		8,4	8,4
	Mgr.MML. Vincenová	3	45,5						0	8,4
31.	Mgr.MMD. Chytilová	3	28,8						0	8,0
32.	Mgr.MMO. Knopp	3	34,3						0	7,7
33.	J. Heller	4	7,5						0	7,5
34.	L. Šajnarová	3	7,3						0	7,3
35.–36.	Bc.MML. Bujnovská	3	12,0						0	7,0
	J. Pelc	3	7,0						0	7,0

Poř.	Jméno	R.	$\sum_{-1}$	Úlohy r1	r2	r3	r4	t3	$\sum_0$	$\sum_1$
37.	J. Šrejbr	1	6,7						0	6,7
38.	T. Dolák	4	6,3						0	6,3
39.	A. Mírková	2	6,0						0	6,0
40.	Bc.[MM]F. Zajíc	4	15,7						0	3,7
41.	Mgr.[MM]S. Lukeš	4	42,4						0	3,2
42.	P. Martínek	2	3,1						0	3,1
43.–45.	D. Daubner	2	3,0						0	3,0
	R. Luc	4	7,0						0	3,0
	M. Picek	2	3,0						0	3,0
46.	T. Poláková	3	2,8						0	2,8
47.–48.	M. Machalová	4	4,7						0	2,7
	M. Sejkorová	2	2,7						0	2,7
49.	A. Šebestíková	2	2,0						0	2,0
50.	F. Kmječ	1	1,5						0	1,5
51.	L. Kubacki	4	0,0						0	0

Sloupeček $\sum_{-1}$ je součet všech bodů získaných v našem semináři, $\sum_0$ je součet bodů v aktuální sérii a $\sum_1$ součet všech bodů v tomto ročníku. Tituly uvedené v předchozím textu slouží pouze pro účely M&M

Adresa redakce:

M&M, OVVP, MFF UK
Ke Karlovu 3
121 16 Praha 2

E-mail: mam@matfyz.cz
WWW: http://mam.matfyz.cz