Zápočtový program NPRG005 2013/2014
simulace dataflow počítače
Vojtěch Hudeček

1. **Úvod**
   Program simuluje výpočet dataflow počítače. Jedná se o výpočetní model lišící se od Von Neumannova v několika ohledech. Především vůbec není definováno pořadí, v jakém se budou provádět instrukce. Program pro tento počítač může být chápan jako cyklická hradlová síť. Každé hradlo má právě dva vstupy a alespoň jeden výstup. Hradlo je schopno provést jednu definovanou operaci, což učiní v okamžiku, kdy jsou oba jeho vstupy ve stavu, kdy je možno je zpracovat. Výsledek operace je distribuován na výstupy hradla, což může případně umožnit zpracování dalšího hradla atd.

2. **Detailnější popis**
   Základními kameny implementace jsou tzv. registry. Registr obsahuje hodnotu, která je použita pro výpočet instrukčních a kontrolních buněk napojených na tento registr. Každá z těchto buněk má ke vstupu napojeny dva registry a výstup distribuuje do dalších. Pro jednodušší a úspornější zápis programů je umožněno, aby jeden registr byl připojen ke vstupům více buněk.
   Aby výpočet buňky proběhl, musí být oba její registry ve stavu enabled.
   Registr má následující tvar:

   R num val type s1 s2 out enabled

   num je číslo registru, *val* je jeho hodnota, *type* udává typ, *s1, s2* nesou informaci o stavu. o*ut* je True, pokud je registr výstupní, podobně *enabled* je True pro povolený registr.
   Každý registr nese kromě hodnoty informaci o stavu. Tato informace je dvousložková, nesou jí položky *s1, s2*. Když do registru dorazí hodnota, přepne se druhá složka (*s2*) do stavu "on". Pokud dorazí informace od řídící buňky, přepne se první složka (*s1*) do stavu True/False. To, jestli je registr povolen se řídí několika pravidly, která se odvíjí od typu daného registru. Registr typu "No" je povolen v okamžiku, kdy dorazí hodnota, nezávisle na první složce. Registr typu "True" (resp. "False") se povolí v okamžiku, kdy obsahuje hodnotu a v první složce odpovídající stav. Pokud se stav změní, registr se zakáže. Registr typu "In" na začátku načte vstup a poté se chová jako typ "True".
   Registry se tedy střídavě povolují a zakazují, podle toho, jaké hodnoty do nich dorazí. Podle toho také probíhají výpočty buněk. Registr může být označen jako výstupní. Program skončí v okamžiku, kdy jsou povoleny všechny výstupní registry.
   Buňky jsou dvou typů – instrukční a kontrolní. Každá buňka obsahuje adresy svých dvou operandů a seznam adres, do kterých má doručit výsledek. Instrukční buňky provádí operace plus, minus, krát a identita. Identita má na obou vstupech stejný registr. Kontrolní buňky provádí porovnání, tzn. větší než (nebo rovno), menší než (nebo rovno) a (ne)rovnost.

3. **Ukázkový program**
   Pro ilustraci popíšu krátký program počítající faktoriál z čísla na vstupu.
   Nejprve se načte vstup do r1; r1 je povolen, je typu "In" a není výstupní.

```
In 1 Zadejte cislo pro vypocet faktorialu:
Register 1 0 In off on False True
```

   r2 slouží pro postupné počítání faktoriálu, registry 3 a 4 mají pomocnou funkci: r3 pro porovnání a r4 pro dekrementaci o 1 , registr 5 obsahuje hodnotu faktoriálu a je výstupní. r5 je typu "False", tedy je povolen v okamžiku, kdy do něj dorazí výsledek False.

```
Register 2 1 No off on False True
Register 3 0 No off on False True
Register 4 1 No off on False True
Register 5 0 Fal off on True False
```

Program pracuje tak, že obsah r2 se vynásobí obsahem r1, uloží se do r2 a r1 se zmenší o jedničku. To provádí následující 2 instrukční buňky:

```
ICell 2
1 2
times
ICell 1
1 4
minus
```

Poslední instrukční buňka kopíruje obsah r2 do r5. Jediná kontrolní buňka v programu zajišťuje, že v okamžiku kdy je r1 == r3 (== 0), povolí se r5 a program skončí.

```
ICell 5
2 2
id
CCell 5
1 3
gt
```

4. Popis implementace

Program je napsán v jazyce Haskell. Pro vnitřní reprezentaci součástí programu slouží datové typy *Register, ICell a CCell*, reprezentující registry a instrukční respektive kontrolní buňky. Zajímavé položky registru jsou *s1, s2 :: StateType a t :: RegType*, což jsou další pomocné (výčtové) typy. Buňky obsahují seznam adres cílů, adresy operandů a položku *func :: InstrFunc* (respektive *fnc :: BoolFunc*), což jsou datové typy zapouzdřující funkce dvou argumentů a řetězec jako slovní popis. Pro jednodušší načtení jsou definovány instance třídy *Read* pro typy *StateType a RegType.* Pro načtení vstupu slouží funkce *parseRegs, parseICells a parseCCells,* které dostanou vstup po řádcích a dojdou-li k příslušné definici, zavolají pomocnou funkci, která vytvoří přísušný registr/buňku. Tyto funkce vrací seznam načtených prvků.
Po spuštění programu je načten vstup. Před načtením registrů se ještě spustí funkce, která získá vstup od uživatele a vrátí seznam zadaných hodnot pro vstupní registry, které se posléze použijí při načítání. Po načtení program vstoupí do hlavního cyklu, který iteruje pomocí *do* bloku volání funkcí aplikujících změny na registry, nejdříve po provedení kontrolních buněk a poté po provedení instrukčních buněk. Po každé aplikaci se nejprve provede aktualizace registrů, tj. nové se povolí, případně zakážou. Poté se zkontroluje, zda zůstávají nějaké nepovolené výstupní registry. Pokud ne, program vypíše hodnoty výstupních registrů a skončí, v opačném případě pokračuje dál.

5. Diskuze

Program poměrně dobře simuluje chování datafow počítače, dají se implementovat i složitější algoritmy. Je zde však velký prostor ke zlepšení. Výhodou reálného dataflow počítače je značná míra paralalelismu vycházející již ze samotného konceptu. Tato možnost v programu zcela chybí. Také by bylo možné rozšířit program o koncept další úrovně paměti, tj. buňky, jak jsou chápany v nynějším programu by tvořily de facto cache, existovala by ale paměť pro ukládání v danou chvíli nepotřebných buněk, které by se tím pádem nezpracovávaly vždy, ale v potřebný okamžik by se přesunuly do cache.