Dokumentace k započtovému programu - Simulátor jednoduchého CPU

Vojtěch Gaďurek

Únor2022

1 Účel programu

Program simuluje smyšlené CPU a to, tak že zpracuje uživatelem zadaný kod a obsah registrů.

2 Psaní vlastního kodu

V základu je hodnota všech registrů nastavena na 0. Pokud chceme změnit počáteční data, můžeme tak učinit pomocí souboru MEMORY.txt. Na každou řádku v souboru píšeme jeden příkaz ve formátu INDEX DATA. Kde INDEX představuje index, kam se daná DATA uloží. Data by mělo být celé kladné čísl. Program data vždy ořízne operací mod 2^8.

Vlastní kod je nutné napsat do souborut CODE.txt. Na každém řádku musí být jeden příkaz ve formátu PŘÍKAZ FIRST SECOND. Všechny příkazy kromě LDA a LDP, STR pracují pouze mezi akumulátory. Řádek kodu tedy může vypadat takto. AND A B Následující příkaz provedl binární AND na obsah registrů A a B a uložil hodnotu výsledku do A.

CPU má následující příkazy.

3 Příkazy

3.1 Unární příkazy

LDN, N0T, SHL, SHR, SCL, SCR, JMP, MOV, MIV,

Mají následující akumulátory A, B, C, D, E, F, G, H s délkou slova 8 bitů

LDN - Načte hodnotu v pořadí za příkazem do akumulátoru určeného ve FIRST

NOT - Provede logický NOT na hodnotu uloženou v akumulátoru urečeného FIRST a výsledek uloží do něj.

SHL - Provede logický posun do leva na dáném akumulátoru

SHR - Provede logický posun do prava na dáném akumulátoru

SCL - Provede logický posun do leva, ale pokud hodnota byla lichá, bude i po operaci

SCR - Provede logický posun do leva, ale byla by hodnota MSB bitu 1, bude i po operaci.

JMP - Nastaví hodnotu pointru na hodnotu akumulátoru

MOV - Nastaví hodnotu akumulátoru na hodnotu registru s indexem 0

MIV - Nastaví hodnotu registru s indexem 0 na hodnotu akumlátoru

3.2 Binární příkazy

ADD, ADC, AN0, XOR, JIF, STR, LDP, ORR,

Mají následujicí akumulátory A, B, C, D s délkou 8 bitů a specialní akumulátor CARRY s délkou 1 bit.

ADD - Sečete hodnoty v akumulátorech a s carry uloží a do FISRT.

ADC - Vymaže hodnotu na carry a sečte hodnoty v akumulátorech a uloží do FIRST

AND - Provede logický AND a výslednou hodnotu uloží do FISRT.

XOR - Provede binární XOR a výslednou hodnotu uloží do FIRST.

JIF - Pokud se hodnota ve FISRT nerovná 0, pak hodnota pointeru se nastaví na hodnotu v SECOND

STR - Hodnota v FIRST se uloží do registru s indexem SECOND.

LDP - Načte hodnotu z registru s indexem SECOND do registru FIRST

ORR - Provede logický ORR a výslednou hodnotu uloží do FIRST.

3.3 Speciální příkazy

end

END - Ukončí program, jeho číselná hodnota je 255.

4 Ovládání debuggeru

Program má v sobě jednoduchý debugger, který umožnuje sledovat jak jsou jednotlivé příkazy vykonávány či si vypsat obsah registrů. Program má následujicí příkazy.

SHOWPRIKAZY Zapne ukazovaní jednotlivých příkazů. Které zobrazí ve formátu PŘÍKAZ FIRST SECOND / HODNOTAFISRT HODNOTASECOND / POINTER

AKUMULATORY Ukáže obsah akumulátorů.

AK_ZMENA Po každém příkazu, ukáže obsah akumulátorů

DEBUG Je stejné, jako zadání SHOWPRIKAZY a poté AK_ZMENA

MEMORY N M Ukáže obsah paměti mezi N a M, pokud není N a M zadáno ukáže všechny registry.

STEP Provede právě jeden příkaz

RYCHLOST N Nastaví délku jednoho tiku na N.

RUN Spustí uživatelův program

STOP / enter Zastaví uživatelův program

END Ukončí debugger

5 Architektura

Program se skládá z dvou hlavních částí a to třídy Device, která simuluje samotné CPU, a funkce runner, která vytvaří debugger nad samotnou simulací cpu.

5.1 Device

Device je potomkem třídy Translator, která se stará o překlad příklazů do čísel, a třídy Operations, která obsahuje samotné příkazy. Pro přidání nového příkazu, je nutné přidat zápis do self.unary nebo self.binary v třídě Translator.

5.2 runner(location)

Umožnuje ovládat uživatelův program, location umožnuje uživateli určit přesnou lokaci souborů CODE.txt a MEMORY.txt.