

Procesor s jednoduchou instrukční sadou

Datum zadání: 21.10.2019

Datum a forma odevzdání: do 20.12.2019 23:59, POUZE přes IS FIT, 4 soubory

Počet bodů: max. 23 bodů

Poznámka: součástí zadání je archiv INP-proj2.zip

Dotazy ohledně projektu: v případě nejasností využijte osobní konzultace (Zdeněk Vašíček, L307)

1 Úvod

Cílem tohoto projektu je implementovat pomocí VHDL procesor, který bude schopen vykonávat program napsaný v rozšířené verzi jazyka BrainF*ck [5]. Ačkoliv jazyk BrainF*ck používá pouze osm jednoduchých příkazů (instrukcí), jedná se o výpočetně úplnou sadu, pomocí které je možné popsat libovolný algoritmus. Na ověření korektní funkce poslouží několik testovacích programů (výpis textu, výpis prvočísel, rozklad čísla na prvočísla, apod.).

Činnost procesoru

Jazyk používá příkazy kódované pomocí tisknutelných 8-bitových znaků. Implementovaný procesor bude zpracovávat přímo tyto znaky (tzn. operační kód procesoru bude sestávat vždy z osmi bitů). Pro jednoduchost budeme uvažovat pouze 11 příkazů uvedených v tabulce níže. Program sestává ze sekvence těchto příkazů. Neznámé příkazy jsou ignorovány, což umožňuje vkládat komentáře přímo do programu. Vykonávání programu začíná první instrukcí a končí jakmile je detekován konec sekvence (znak s ASCII hodnotou 0). Program i data jsou uložena ve stejné paměti mající kapacitu 8192 8-bitových položek. Program je uložen od adresy 0 a je vykonáván nelineárně (tzn. obsahuje skoky). Data jsou uložena od adresy 0x1000 (tj. 4096 desítkově). Obsah paměti nechť je pro jednoduchost inicializován na hodnotu nula. Pro přístup do paměti se používá ukazatel (ptr), který je možné přesouvat o pozici doleva či doprava. Paměť je chápána jako kruhový buffer uchováající 8-bitová čísla bez znaménka. Posun doleva z adresy 0x1000 tedy znamená přesun ukazatele na konec paměti odpovídající adrese 0x1FFF.

Implementovaný procesor nechť podporuje příkazy definované v následující tabulce. Operační kódy, které se v tabulce nenacházejí jsou procesorem ignorovány.

příkaz	operační kód	význam	ekvivalent v C
>	0x3E	inkrementace hodnoty ukazatele	ptr += 1;
<	0x3C	dekrementace hodnoty ukazatele	ptr -= 1;
+	0x2B	inkrementace hodnoty aktuální buňky	*ptr += 1;
-	0x2D	dekrementace hodnoty aktuální buňky	*ptr -= 1;
[0x5B	je-li hodnota aktuální buňky nulová, skoč za odpovídající příkaz] jinak pokračuj následujícím znakem	while (*ptr) {
]	0x5D	je-li hodnota aktuální buňky nenulová, skoč za odpovídající příkaz [jinak pokračuj následujícím znakem	}
.	0x2E	vytiskni hodnotu aktuální buňky	putchar(*ptr);
,	0x2C	načti hodnotu a ulož ji do aktuální buňky	*ptr = getchar();
\$	0x24	ulož hodnotu buňky do pomocné proměnné	tmp = *ptr;
!	0x21	hodnotu pomocné proměnné ulož do aktuální buňky	*ptr = tmp;
null	0x00	zastav vykonávání programu	return;

V případě příkazů [a] manipulujících s ukazatelem do programového kódu (instrukčním čítačem PC) je zapotřebí detekovat odpovídající pravou, respektive levou, závorku. Možností je několik, nejjednodušší je postupně inkrementovat (respektive dekrementovat) ukazatel a počítat počet závorek (viz dále). Pomocná proměnná tmp je uložena v paměti na adrese 0x1000. Ukazatel ptr ukazuje po resetu na adresu 0x1000.

Mikrokontroler

Aby bylo možné vykonávat smysluplný program, je procesor připojen k paměti programu a dat o celkové kapacitě 8kB a vstupně-výstupnímu rozhraní umožňující načítat a vypisovat data. Výsledný mikrokontroler je součástí zip archivu jakožto projekt pro FITkit.

Vstup dat je řešen pomocí maticové klávesnice. Jakmile procesor narazí na instrukci načtení hodnoty (operační kód 0x2C), vykonávání se pozastaví do té doby, než je stisknuto některé z tlačítek klávesnice. Tlačítka 0-9 jsou interpretována jako znaky '0' až '9' s ASCII hodnotami 48 až 57. Tlačítko * a # je interpretováno jako konec řádku s ASCII hodnotu 10.

Výstup dat je řešen pomocí LCD displeje, posun kurzoru na displeji je řešen automaticky. Při zápisu většího počtu znaků, než-li dovoluje kapacita aktivní části displeje, dojde k návratu na první znak a dříve zapsané znaky se postupně přepisují. Pozor. Podle verze FITkitu může být nutné v souboru *top.vhd* upravit hodnotu parametru LCD2x16 (viz popis v souboru).

2 Úkoly

1. Obsah souboru *login.b* obsahujícího program v jazyce BrainF*ck, který tiskne řetězec *xlogin01*, zkopírujte do debuggeru na adrese [4]. Tlačítkem "Start debugger" spustíte krokování a sledujte, co způsobuje která instrukce, jak se pohybuje programový čítač a ukazatel do paměti dat. Vytvořte program, který vytiskne na displej *Váš login* (na velikosti písmen nezáleží). Snažte se v programu využít všechny dostupné příkazy s výjimkou příkazu načtení. Pokuste se vytvořit co nejkratší program tisknoucí *Váš login*.
2. Seznamte se s kódem v souborech *ram.vhd* (paměť programu a dat), *cpu.vhd* (rozhraní procesoru) a *top.vhd* (strukturní popis mikrokontroleru). Povšimněte si, kde je definován program v jazyce BrainF*ck a jak jej lze modifikovat.
3. Do souboru *cpu.vhd* doplňte vlastní VHDL kód, který bude syntetizovatelným způsobem popisovat implementaci procesoru vykonávajícího program zapsaný v jazyce BrainF*ck.

Rozhraní procesoru je pevně dané a skládá se z čtyř skupin signálů: synchronizace, rozhraní pro paměť programu a dat, vstupní rozhraní a výstupní rozhraní.

Synchronizační rozhraní tvoří tři signály. **CLK** - hodinový synchronizační signál. Procesor pracuje vždy při vzestupné hraně hodinového signálu. **RESET** - asynchronní nulovací signál. Je-li **RESET=1**, procesor ziniculuje svůj stav (**PTR=0, PC=0**). **EN** - povolení činnosti procesoru. Pokud je signál **RESET** uvolněn (**RESET=0**) a **EN=1**, procesor postupně začne vykonávat program od adresy 0.

Rozhraní *synchronní* paměti, která slouží k uchování programu a dat je tvořeno třemi datovými a dvěma řídicími signály. Signál **DATA_ADDR** o šířce 13 bitů slouží k adresaci konkrétní buňky paměti. Signál **DATA_RDATA** (načtená data) obsahuje 8-bitovou hodnotu buňky na adrese **DATA_ADDR**. Signál **DATA_WDATA** (zapisovaná data) nechť obsahuje 8-bitovou hodnotu, kterou se má přepsat buňka na adrese **DATA_ADDR**. Rozhraní pracuje následovně. Platí-li **DATA_EN=1** (povolení činnosti paměti) a **DATA_RDWR=0** (volba režimu čtení / zápis), signál **DATA_RDATA** je aktualizován hodnotou buňky na adrese **DATA_ADDR**. Je-li **DATA_EN=1** a **DATA_RDWR=1**, hodnota buňky na adrese **DATA_ADDR** je přepsána hodnotou signálu **DATA_WDATA** a signál **DATA_RDATA** je aktualizován hodnotou **DATA_WDATA**. K aktualizaci hodnoty signálu **DATA_RDATA** dochází pouze pokud **DATA_EN = 1** (tj. aktivním signálu povolení činnosti). Signály **DATA_ADDR**, **DATA_EN** a **DATA_WDATA** jsou čteny a signál **DATA_RDATA** aktualizován při vzestupné hraně hodinového signálu **CLK**.

Vstupní rozhraní, které je připojeno na řadič klávesnice pracuje následovně. Při požadavku na data procesor nastaví signál **IN_REQ** na 1 a čeká tak dlouho, dokud signál **IN_VLD** (input valid) není roven 1. Jakmile se tak stane, může procesor přečíst signál **IN_DATA**, který obsahuje ASCII hodnotu stisknuté klávesy.

Výstupní rozhraní napojené na LCD displej pracuje následovně. Při požadavku na zápis dat procesor nejdříve musí otestovat stav signálu **OUT_BUSY**. Tento signál indikuje, že je LCD displej zaneprázdněn vyřizováním předchozího požadavku. Jakmile je **OUT_BUSY=0**, procesor inicializuje signál **OUT_DATA** zapisovanou ASCII hodnotou a současně na jeden hodinový takt nastaví signál **OUT_WE** (povolení zápisu) na 1.

Činnost procesoru důkladně ověřte pomocí simulace a následně na přípravku FITkit. K ověření použijte dodané testovací programy (viz soubor *top.vhd*) případně použijte vlastní. Předpokládejte, že na vstupu bude vždy validní kód.

4. Do souboru *top.vhd* vložte program vytvořený v prvním bodě zadání vypisující Váš login. Projekt vysyntetizujte (tj. přeložte v prostředí Xilinx ISE) a odsimulujte.

Poznámka: Hlášek během syntézy (info, warning) vzniklých mimo entitu CPU informujících o nezapojených signálech si nevšímejte, nelze je bohužel jednoduše potlačit.

3 Odevzdává se

Do IS FIT se odevzdávají následující **4 soubory** (nikoliv ZIP či jiný archiv). Soubor **login.b** obsahující program v jazyce BrainF*ck vypisující Váš login (jedná se o výsledek bodu 1 zadání). Soubor **cpu.vhd** obsahující implementaci procesoru (jedná se o výsledek bodu 3 zadání). Výsledky 4. bodu zadání, tj. výsledek syntézy nacházející se v souboru *build/fpga/inp.srp* (soubor **inp.srp**), screenshot ze simulace (soubor **inp.png**) zachycující stav signálů kolem okamžiku, kdy se narazí na instrukci HALT (tj. konec programu). Není potřeba uvádět všechny signály detailně, avšak na obrázku by měl být vidět měnící se stav automatu, čitelné signály určující stav automatu procesoru a signál Display zachycující obsah LCD displeje.

4 Hodnocení

Za kompletní implementaci procesoru (tj. splnění bodu 3) lze získat až 17 bodů. Za implementaci procesoru podporujícího pouze jednoduchý while cyklus (tj. nepodporující vnořené while cykly) lze získat až 12 bodů. Za první a poslední bod zadání lze získat až 6 bodů. Odevzdání po termínu je penalizováno bodovou srážkou ve výši 10 bodů a to za každý započatý den.

5 Upozornění

Pracujte samostatně, nikomu nedávejte svoji práci k opsání.

Plagiátorství se hodnotí 0 body, neudělením zápočtu a případným dalším adekvátním postihem dle platného disciplinárního řádu VUT v Brně. Přejmenování proměnných, změna pořadí jednotlivých bloků či modifikace komentářů není považováno za autorské dílo a na řešení bude nahlíženo jako na plagiát.

6 Revize

16.10.2019 - první verze dokumentu

25.10.2019 - doplněn signál 'sel' ovládající MX2 v obrázku 1

04.11.2019 - původně textové odkazy předělány na hypertextové (funguje proklik)

Odkazy

- [1] <http://merlin.fit.vutbr.cz/FITkit/download.html> - image s vývojovými nástroji pro VirtualBox

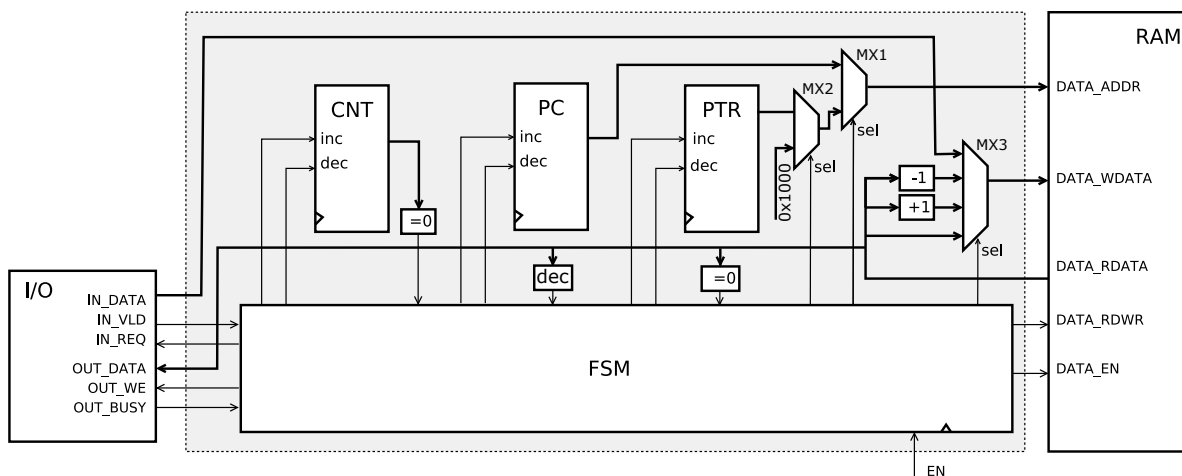
- [2] <http://merlin.fit.vutbr.cz/FITkit/docs/navody/kompilacev2.html> - překladový systém
- [3] <https://esolangs.org/wiki/Brainlove> - popis instrukční sady
- [4] <http://www.fit.vutbr.cz/~vasicek/inp19> - online debugger Brainlove / BrainFuck
- [5] <https://esolangs.org/wiki/Brainfuck> - popis instrukční sady
- [6] <http://www.hevanet.com/cristofd/brainfuck/> - několik programů napsaných v jazyce BrainFuck
- [7] https://esolangs.org/wiki/Brainfuck_algorithms - různé algoritmy

Návod

Následující řádky jsou určeny těm, kteří doposud netuší jak procesor naimplementovat. Obecně platí, že procesor se skládá z datové cesty obsahující registry, ALU, apod. a řídicí cesty obsahující automat. Stejně tak je tomu i v tomto případě. Blokové schema možné implementace je uvedeno na obrázku 1.

Abychom mohli vykonávat program, obsahuje datová cesta tři registry (čítače) s možností inkrementace a dekrementace. Registr PC slouží jako programový čítač (tj. ukazatel do paměti programu), registr PTR jako ukazatel do paměti dat a registr CNT slouží ke korektnímu určení odpovídajícího začátku/konce příkazu while (počítání otevíracích / uzavíracích závorek, viz. popis instrukční sady). Mimo to datová cesta obsahuje multiplexor MX1, pomocí kterého lze během čtení z paměti definovat, zda-li se jedná o adresu programu nebo adresu dat, multiplexor MX2 pro výběr adresy dat a multiplexor MX3, který řídí zapisovanou hodnotu. Zapsat je možné buď hodnotu načtenou ze vstupu, hodnotu v aktuální buňce sniženou o jedničku, hodnotu aktuální buňky zvýšenou o jedničku nebo hodnotu získanou při posledním čtení z paměti. V případě, že se rozhodnete NEimplementovat podporu vnořených while cyklů, registr CNT nepotřebujete.

Všechny řídicí signály jsou ovládány automatem, tak jak je uvedeno ve schematu. Při tvorbě VHDL kódu se inspirujte procesorem probíraným na cvičeních. V prvním kroku implementujte registry (konstrukce process) a multiplexory (dataflow popis) a poté postupujte od jednodušších instrukcí ke složitějším. Implementaci smyček si ponechte až úplně na závěr. Korektní činnost ověřte pomocí simulace a vlastních či přiložených programů (pozor, v simulaci není podpora pro vstup dat z klávesnice).



Obrázek 1: Blokové schema mikrokontroleru

Nevíte-li jak implementovat automat pomocí VHDL, podívejte se do materiálů ke cvičením INP. Jako návod pro implementaci automatu by měl posloužit pseudokód popisující chování jednotlivých instrukcí uvedený v tabulce 1. Máte-li s implementací problémy, vytvořte jednodušší verzi automatu, který nepodporuje vnořené smyčky. Pseudokód je uveden v tabulce 2.

Projekt pro FITkit

Zdrojové soubory využívají komponenty a překladové skripty, které se používají v rámci projektu FITkit. K usnadnění práce doporučujeme využít image pro VirtualBox, který obsahuje všechny potřebné nástroje. Po stažení image z adresy [1] a spuštění v aplikaci VirtualBox nezapomeňte aktivovat licenci pro Xilinx ISE (spuštěním průvodce x1cm).

Po spuštění QDevKitu stáhněte aktuální zdrojové kódy a do podadresáře *apps* rozbalte obsah souboru *INP-proj2.zip*. Při následném spuštění QDevKitu se ve stromu dostupných aplikací objeví položka INP a pod ní tento projekt. Pomocí kontextového menu případně dvojklikem na této položce lze připojený FITkit

naprogramovat případně projekt přeložit či spustit simulaci. Celým procesem lze projít i v příkazové řádce (příkazy `gmake`, `gmake load`, `gmake isim`, `gmake term`) [2]. Před použitím příkazu `gmake` je zapotřebí vytvořit Makefile pomocí příkazu `fcmake`.

Součástí zadání je vysyntetizované vzorové řešení (soubor *build/inp.bin*) obsahující program, který po spuštění očekává na vstupu číslo (tj. sekvence číslic ukončená stiskem klávesy #), které je následně rozloženo na prvočísla. Pokud FITKit naprogramujete (pozor, nesmíte projekt přeložit) dodaným kódem a spustíte terminál, procesor bude čekat na stisk klávesy. Tento stav je indikován svitem LED diody D4. Vzorové řešení je určeno pro FITkit s dvouřádkovým displejem (tzn. verze 2.x). Pro ověření funkce na FITkitu s jednořádkovým displejem je zapotřebí nakopírovat obsah adresáře *build/fitkit.v1* do adresáře *build* jinak v důsledku nekompatibility nebudou vidět znaky na druhé polovině displeje.

Projekt je možné **simulovat** pomocí přiloženého test bench souboru (*fpga/sim/tb.vhd*) a simulátoru Xilinx ISIM. Simulaci lze vyvolat z aplikace QDevKit případně zadáním příkazu `gmake isim` v adresáři s projektem. Skript starající se o vložení signálů do simulace a její spuštění je umístěn v souboru *fpga/sim/isim.tcl*. Skript lze modifikovat dle potřeb. Test bench (VHDL kód v souboru *tb.vhd*) obsahuje FPGA entitu, generátor hodinového a resetovacího signálu a emulátor LCD displeje.

příkaz / stav	pseudokód
výchozí stav	$PC \leftarrow 0, PTR \leftarrow 0x1000, CNT \leftarrow 0, TMP \leftarrow 0x1000$
>	$PTR \leftarrow 0x1000 + (PTR + 1) \% 0x1000, PC \leftarrow PC + 1$
<	$PTR \leftarrow 0x1000 + (PTR - 1) \% 0x1000, PC \leftarrow PC + 1$
+	$DATA_RDATA \leftarrow mem[PTR]$ $mem[PTR] \leftarrow DATA_RDATA + 1, PC \leftarrow PC + 1$
-	$DATA_RDATA \leftarrow mem[PTR]$ $mem[PTR] \leftarrow DATA_RDATA - 1, PC \leftarrow PC + 1$
.	while (OUT_BUSY) {} $OUT_DATA \leftarrow mem[PTR], PC \leftarrow PC + 1$
,	$IN_REQ \leftarrow 1$ while (!IN_VLD) {} $mem[PTR] \leftarrow IN_DATA, PC \leftarrow PC + 1$
[$PC \leftarrow PC + 1$ if ($mem[PTR] == 0$) $CNT \leftarrow 1$ while ($CNT != 0$) $c \leftarrow mem[PC]$ if ($c == '['$) $CNT \leftarrow CNT + 1$ elsif ($c == ']'$) $CNT \leftarrow CNT - 1$ $PC \leftarrow PC + 1$
]	if ($mem[PTR] == 0$) $PC \leftarrow PC + 1$ else $CNT \leftarrow 1, PC \leftarrow PC - 1$ while ($CNT != 0$) $c \leftarrow mem[PC]$ if ($c == ']'$) $CNT \leftarrow CNT + 1$ elsif ($c == '['$) $CNT \leftarrow CNT - 1$ if ($CNT == 0$) $PC \leftarrow PC + 1$ else $PC \leftarrow PC - 1$
\$	$DATA_RDATA \leftarrow mem[TMP]$ $mem[TMP] \leftarrow DATA_RDATA, PC \leftarrow PC + 1$
!	$mem[PTR] \leftarrow mem[TMP], PC \leftarrow PC + 1$
null	$PC \leftarrow PC$
ostatní	$PC \leftarrow PC + 1$

Tabulka 1: Popis chování (pseudokód) jednotlivých instrukcí procesoru.

příkaz / stav	pseudokód
[$PC \leftarrow PC + 1$ if (mem[PTR] == 0) do $c \leftarrow \text{mem}[PC]$ $PC \leftarrow PC + 1$ until (c == ']')
]	if (mem[PTR] != 0) do $PC \leftarrow PC - 1$ $c \leftarrow \text{mem}[PC]$ until (c == '[') $PC \leftarrow PC + 1$

Tabulka 2: Zjednodušená implementace instrukcí procesoru nepodporující vnořené smyčky