

Michał Stochmiałek <misto@pld-linux.org>

Tomasz Poradowski <batonik@batonik.net>

Dokumentacja projektu

**SmArc**

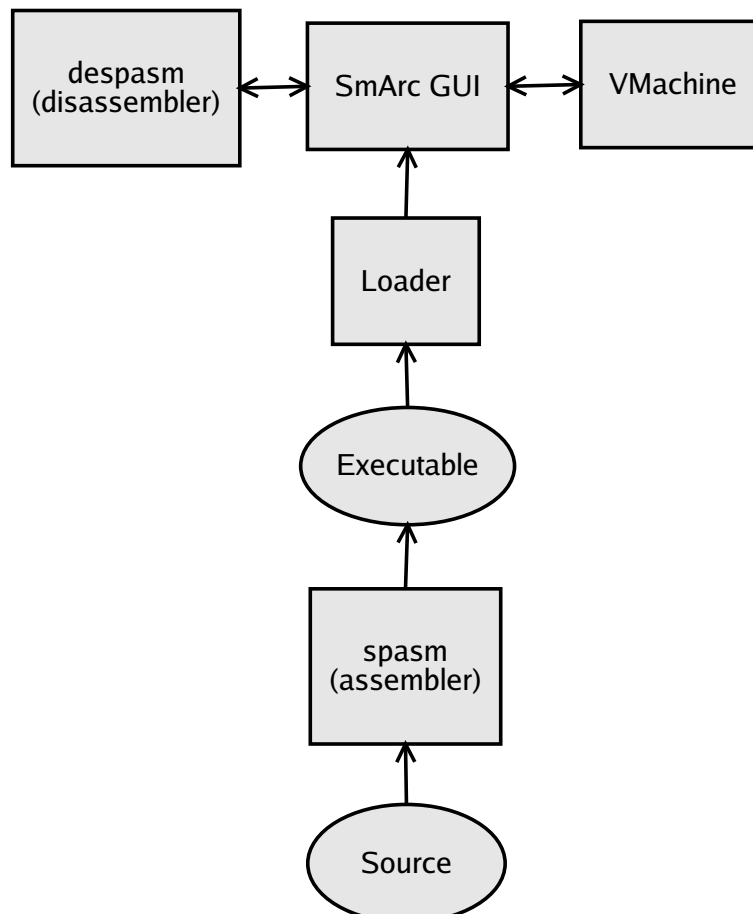
## 1. Wstęp

Wirtualna maszyna **SmArc** powstała jako projekt akademicki, mający na celu bliższe poznanie architektury oraz sposobu działania procesora SPARC. **SmArc** (od *Small SPARC*) zawiera jego wersję uproszczoną, jednakże wiernie oddaje ona realia i większość aspektów projektowych oraz technologicznych zastosowanych w oryginale (więcej informacji w specyfikacji SPARCV8[1]).

## 2. Elementy SmArc

W skład pakietu **SmArc** wchodzi następujące składniki (szczegółowy opis znajduje się w dalszej części):

- *vmachine* - wirtualna maszyna oparta na uproszczonej wersji procesora SPARC
- *spasm* - assembler programów napisanych w języku **SmArcAs** (uproszczonej wersji assemblera SPARC)
- *despasm* - deassembler plików wykonywalnych zapisanych w formacie maszyny wirtualnej **SmArc**
- *smarcgi* - interfejs graficzny wirtualnej maszyny **SmArc**
- loader - moduł służący do wczytania plików wykonywalnych **SmArc**
- examples - przykładowe programy źródłowe w języku **SmArcAs**



## 2.1. vmachine

(wstęp - co to jest za moduł, co robi, gdzie się z niego korzysta)

### 2.1.1. Założenia

(uproszczony (z mniejszą liczbą rozkazów) procesor SPARC, 5-cio potokowy, forwarding, stall; brak: annul bit, koprocatora, połowy PSR, etc.)

### 2.1.2. Realizacja

(z czego się składa, jakie rejestry, co zawierają, jaka pamięć i ile jej jest) (okna rejestrów - jak się zmieniają) (loader binarek)

Funkcje realizowane w wirtualnej maszynie **SmArc** są zgodne z opisem zawartym w SPARC manual [1].

## 2.2. spasm

Program *spasm* służy do tłumaczenia plików źródłowych napisanych w języku **SmArcAs** (uproszczonej wersji assemblera SPARC) do kodu wykonywalnego (binarnego) w formacie maszyny wirtualnej **SmArc**. *spasm* jest uruchamiany z linii poleceń:

```
spasm [-h] [-v] [-q] [-o <plikbin>] <plikasm>
```

gdzie:

<plikasm>	plik źródłowy zawierający program w języku <b>SmArcAs</b> .
-h	wyświetla krótką informację o składni wywołania programu
-v	włącza tryb <i>debug</i> - program będzie wtedy wyświetlał sporą ilość informacji dotyczących analizy składniowo-leksykalnej; nie zostaną także usunięte osobne pliki tymczasowe programu zawierające sekcje danych oraz kodu
-q	wyłącza informacje ostrzegawcze (o braku niektórych elementów składni języka, których brak nie powoduje jednakże błędów przy generowaniu pliku wynikowego) (użycie NIE ZALECANE)
-o <plikbin>	określa nazwę pliku wynikowego, do którego ma zostać zapisany program wykonywalny (<plikbin>); jeżeli parametr ten zostanie pominięty, to zostanie użyta domyślna nazwa pliku wynikowego 'spasm.output'

Jeżeli podczas analizy pliku źródłowego nie wystąpiły żadne błędy, *spasm* nie wypisze żadnego komunikatu (zachowanie podobne do standardowych kompilatorów) oraz utworzy plik wykonywalny w formacie maszyny wirtualnej (opis poniżej); w przeciwnym przypadku na ekranie pojawią się komunikaty błędów i plik wynikowy nie zostanie utworzony (lub będzie pusty).

## 2.3. despasm

Program *despasm* to prosty deassembler kodu binarnego wygenerowanego za pomocą assemblera *spasm*. *despasm* jest domyślnie wywoływany bezpośrednio z interfejsu graficznego *smarc-gui*. Jest jednak możliwe jego ręczne wywołanie:

```
despasm [-b] [-n] <prog>
```

gdzie:

- <prog>** plik binarny w formacie wykonywalnym maszyny wirtualnej **SmArc**.
- b** powoduje, że pomijane jest wyświetlanie adresu danego rozkazu (liczonego od początku, czyli adresu 0x0) na początku każdej linii
- n** modyfikuje domyślny tryb wyświetlania nazw rejestrów w argumencie rozkazu z zapisu **%r<N>** do bardziej przyjaznego programiście zapisu alternatywnego - **%g<M>**, **%o<M>**, **%i<M>**, **%l<M>**, **%sp** oraz **%fp** (gdzie  $0 \leq N \leq 31$ ,  $0 \leq M \leq 7$ ).

Domyślnie wyświetla on dla każdego słowa rozkazowego na standardowym wyjściu linię postaci:

```
adres:   rozkaz   argumenty
```

## 2.4. smarcgui

*smarcgui* to graficzny interfejs maszyny wirtualnej **SmArc**. Intuicyjna realizacja zapewnia łatwy i zorganizowany dostęp do wewnętrznych elementów maszyny wirtualnej, takich jak rejestry i pamięć, oraz do stanów pośrednich w poszczególnych krokach przetwarzania potokowego.

## 2.5. examples

W katalogu "examples" zawarte zostały poglądowe programy mające na celu pokazanie jednej lub więcej właściwości procesora SPARC. Niektóre z nich to zwykłe, typowe programy testowe.

- addloop.asm - pętla wykonywana zadaną ilość razy
- arith.asm - instrukcje arytmetyczne
- forward.asm - demonstracja "forwardingu" w procesorze SPARC
- load.asm - odczyt i zapis z/do pamięci DCache
- logic.asm - operacje logiczne na rejestrach
- loop.asm - instrukcje branch
- memcpy.asm - kopiowanie danych
- procedure.asm - instrukcja call wywołanie fragmentu programu (procedury)
- recurrence.asm - instrukcje call, save, restore rekurencyjne wywołanie fragmentu programu (procedury)
- stall.asm - demonstracja wystąpienia "stall" w procesorze SPARC

# 3. Założenia i szczegóły techniczne

## 3.1. Założenia języka SmArcAs

Składnia języka **SmArcAs** (*Small SPARC Assembler*):

```
program ::= [linia]*
linia   ::= [wyrażenie] [!komentarz]
wyrażenie ::= [etykieta:] [instrukcja]
instrukcja ::= rozkaz | pseudorozkaz | dyrektywa
```

gdzie:

- [ ] - element opcjonalny
- \*
- | - alternatywa

Rozpoznawane przez *spasm* rozkazy obejmują (liczby w nawiasie oznaczają składnię rozkazu/grupy rozkazów - opisane dalej):

- rozkazy arytmetyczne (1):  
add, addx, addcc, addxcc, sub, subx, subcc, subxcc
- rozkazy logiczne (1):  
and, andcc, or, orcc, xor, xorcc, andn, andcc, orn, orncc, xorn, xorncc
- rozkazy przesunięć (1):  
sll, srl, sra
- rozkazy skoków warunkowych branch (2) (nawiasach [ ] nazwy alternatywne):  
bn, bleu, ba, bgu, be [bz], bcs [blu], bne [bnz], bcc [bgeu] ble, bneg, bg, bpos, bl, bvs, bge, bvc
- rozkazy odczytu(3) / zapisu(4) z pamięci:  
ld, ldub, ldsb, ldub, ldsh, st, stb, sth, std
- rozkazy sterujące:  
call (5), jmp1 (6)
- inne rozkazy:  
sethi (7), save (1), restore (1)

### Składnia rozkazów:

1. *instr* <src1>, <src2>/<imm13>, <dst>
2. *branch* <etykieta>
3. *load* [<adres>], <dst>
4. *store* <dst>, [<adres>]
5. *call* <etykieta>
6. *jmp1* <adres>, <dst>
7. *sethi* <const>, <dst>

gdzie:

- <src1> - pierwszy rejestr źródłowy
- <src2> - drugi rejestr źródłowy
- <imm13> - stała 13 bitowa
- <dst> - rejestr docelowy wyniku
- <etykieta> - etykieta (adres) w sekcji kodu
- <const> - stała liczbowa (lub adres wskazywany przez etykietę danych/kodu)
- <adres> - kombinacja kilku powyższych składników:
  - <src1> + <src2>
  - <src1>
  - <src1> + <const>
  - <src1> - <const>
  - <const>

Dokładniejsze informacje dot. składni, formatu słów rozkazowych i kodowania rozkazów można znaleźć w specyfikacji SPARCV8 [1]. Tam również znajdują się rozkazy procesora SPARC, których implementacja **SmArc** nie obejmuje (inne rozkazy arytmetyczne, rozkazy korzystające z koprocessora, itp.)

Pseudorozkazy rozpoznawane przez *spasm* oraz ich interpretacja znajdują się w poniższej tabeli:

Pseudorozkaz	Rozkaz(y)
mov %src, %dst	or %g0, %src, %dst
cmp %reg1, %reg2	subcc %reg2, %reg1, %g0
tst %reg	orcc %g0, %reg, %g0
not %reg	xnor %g0, %reg, %reg
neg %reg	sub %g0, %reg, %reg
inc %reg	add %reg, 1, %reg
dec %reg	sub %reg, 1, %reg
clr %reg	or %g0, %g0, %reg
restore	restore %g0, %g0, %g0
save	save %g0, %g0, %g0
ret	jmp1 %i7 + 8, %g0
jmp <etykieta>	jmp1 <etykieta>, %g0
skipz	bnz . + 8 <sup>1 2</sup>
skipnz	bz . + 8 <sup>1 2</sup>
set <const>, %reg	sethi %hi(<const>), %reg
	or %reg, %lo(<const>), %reg
	lub
	or %reg, <const>, %reg
	(gdy <const> ≤ <imm13>)
nop	sethi 0, %g0

### 3.2. Dyrektywy SmArcAs

Postać dyrektywy	Funkcja
.text	Początek sekcji programu (rozkazów)
.data	Początek sekcji danych
.align k	Wyrównanie adresu do granicy k bajtów (k = 2 lub 4)
.skip n	Rezerwacja n kolejnych bajtów
.byte w1,w2,w3,...	Nadanie wartości w1, w2, .. kolejnym bajtom
.half w1,w2,w3,...	Nadanie wartości w1, w2, .. kolejnym półsłowom
.word w1,w2,w3,...	Nadanie wartości w1, w2, .. kolejnym słowom
.ascii 'znaki...'	Umieszczenie w kolejnych bajtach kodów znaków ASCII <sup>3</sup>
=	Przypisanie nazwie (lewostronnej) wartości (prawostronnej)

Inne założenia:

- małe i duże litery są rozróżniane za wyjątkiem nazw symboli specjalnych,
- rozkazy są pisane wyłącznie małymi literami,
- symbole specjalne zaczynają się od znaku procentu (%)
- etykiety i nazwy zaczynają się od litery; po etykietce musi być dwukropek (:).
- liczby mogą być w zapisie dziesiętnym lub szesnastkowym; w zapisie szesnastkowym liczby muszą być poprzedzone znakiem zera i litery 'x' (np. 0xffff),
- łańcuch znaków umieszczany jest w pojedynczym cudzysłowie ('),

<sup>1</sup> Kropka (.) w polu argumentu oznacza adres bieżącego rozkazu;

<sup>2</sup> **SmArc** nie obsługuje *annul bit*, więc generowane rozkazy nie ustawiają tego bitu(!)

<sup>3</sup> Znaki sterujące zapisuje się następująco: \b - backspace, \n - new line, \r - carriage return, \t - horizontal tab.

- dostępne operatory unarne:
  - `%lo(<const>)` - wydziela 10 mniej znaczących bitów stałej (32-bitowej)
  - `%hi(<const>)` - wydziela 22 bardziej znaczących bitów stałej (32-bitowej)
- nazwy rejestrów mogą być zapisywane w formie `%r<N>` lub alternatywnie jako `%g<M>`, `%o<M>`, `%i<M>`, `%l<M>`, `%sp` oraz `%fp` (gdzie  $0 \leq N \leq 31$ ,  $0 \leq M \leq 7$ ).

### 3.3. Format pliku wykonywalnego maszyny wirtualnej

Asembler *spasm* scala część danych oraz część kodu programu do pliku wynikowego o następującym formacie:

Pole	Długość	Typ	Zawartość
NAGŁÓWEK	4 bajty	ciąg ASCII	stały ciąg 'SMARC' identyfikujący plik wykonywalny
WERSJA	1 bajt	cyfra 0-9 w kodzie ASCII	numer wersji pliku wykonywalnego (obecnie jedyna obsługiwana wersja to '0')
ID_DANYCH	4 bajty	ciąg ASCII	stały ciąg 'DATA' zapowiadający sekcję danych
DL_DANYCH	4 bajty	słowo 32-bitowe (big-endian)	zawiera długość bloku danych (w bajtach) wyrównanego do granicy słowa 32-bitowego
DANE	DL_DANYCH bajtów	sekcja danych	
ID_KODU	5 bajtów	ciąg ASCII	stały ciąg 'INSTR' zapowiadający sekcję kodu
DL_KODU	4 bajty	słowo 32-bitowe (big-endian)	zawiera długość bloku kodu (w bajtach)
KOD	DL_KODU bajtów	sekcja kodu	

## 4. Instalacja

### 4.1. Wymagania

flex-devel, bison-devel, stdc++-devel, Gtk-devel, glade-devel, etc.

### 4.2. Budowanie pakietu SmArc

```
$ tar xzf smarc-XX.tar.gz
$ cd smarc
$ make
```

## Bibliografia

- [1] SPARCV8 (32-Bit SPARC) Architecture Book, <http://www.sparc.org/standards.html>