ćwiczenie 3

Zaawansowane architektury komputerowe

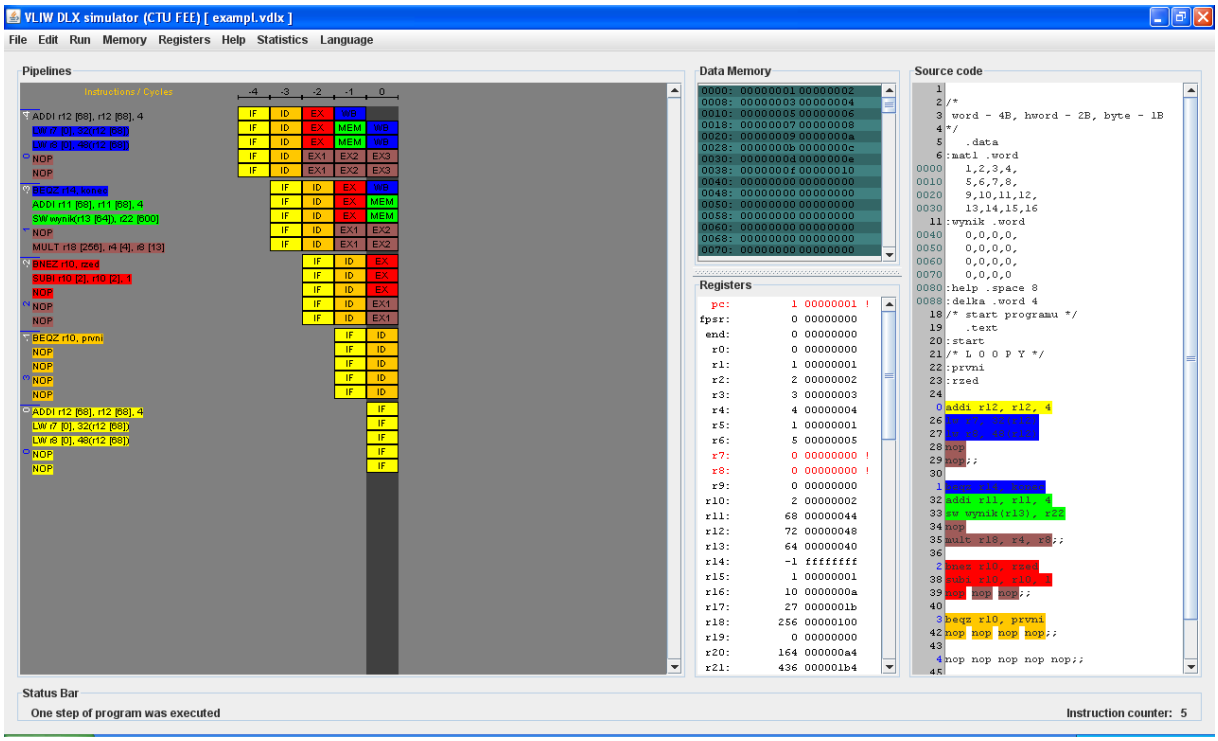
Procesory o architekturze VLIW/DLX

1 Wprowadzenie

Architektura VLIW (Very Long Instruction Word) jest przykładem poszukiwania rozwiązań przyśpieszających działanie procesora (ILP) oraz zmiany sposobu konstruowania programu. Bieżące ćwiczenie laboratoryjne pozwoli na zapoznanie się ze sposobem konstruowania oprogramowania dla takich architektur. Z jednej strony student zapozna się ze sposobem pracy narzędzi dla takiej architektury – ich ograniczeniami, z innej strony zapozna się z odmiennym podejściem do wykonywania programu w takiej architekturze.
Architektura zawiera ten sam zestaw ISA, poznany na poprzednich ćwiczeniach, a co z tym się wiąże ćwiczenie wykonywane jest ze świadomością istnienia mikrooperacji (w tym ćwiczeniu są one ukryte) również w tym przypadku.

1.1 Przygotowanie ćwiczenia

- Pobierz materiały ze strony internetowej: .zip
- Rozpakuj materiały w katalogu roboczym
- Uruchom symulator (najczęściej Enter na vliw_dlx.jar, ewentualnie z linii komend)



Rys. 1. Główne okno symulatora procesora o architekturze VLIW

Sposób programowania procesora polega na grupowaniu instrukcji po 5, zakończonych dwoma średnikami, co odpowiada ilości potoków. Edycja programu odbywa się bezpośrednio w oknie edycji. Dodatkowo można wykorzystywać etykiety:
:mat1 .word

1,2,3,4,
5,6,7,8,
9,10,11,12,
13,14,15,16

Pozostałe elementy syntaktyki programu można prześledzić na podstawie Rys.1. Miarą efektywności wykonanego programu jest ilość wykorzystanych instrukcji NOP, oraz ilość cykli zegarowych.

Instrukcje dostępne w każdym potoku zebrane są w postaci poniższych tabel (z każdym potokiem związana jest jednostka wykonawcza) :

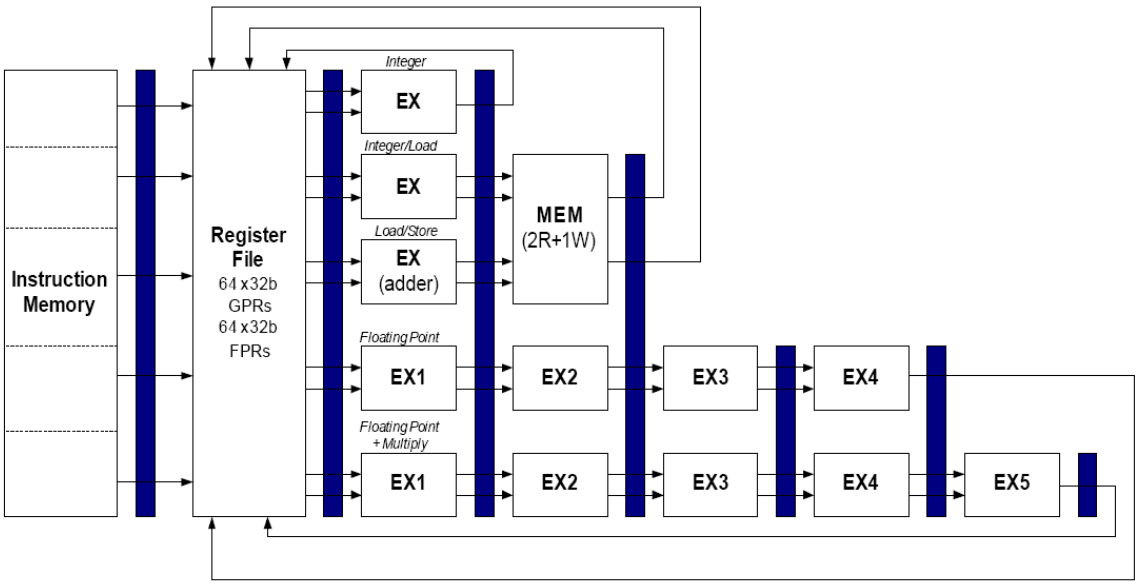
Tabela 1 Potoki, zwłoka dla każdej jednostki wykonawczej

VLIW pipeline	***Operation group allowed***	***Latency***
1	Control, ALU	2
2	ALU, Load	3
3	Load / Store	3
4	Float	4
5	Float, Multiply	5

Tabela 2 Grupy operacji

Operation group	***DLX instructions = VLIW DLX operations***
Control	beqz, bnez, j, jal, jr, jalr, bfpt, bfpf
Float	addf, subf, cvtf2i, cvti2f, eqf, nef, ltf, gtf, lef, gef
Multiply	multf, mult, multu
ALU	add, addu, addi, addui, sub, subu, subi, subui, and, andi, movf, lhi, or, ori, xor, xori, sll, slli, srl, srli, sra, srai,seq, sne, slt, sgt, sle, sge, seqi, snei, slti, sgti, slei, sgei, sequ, sneu,sltu, sgtu, sleu, sgeu, sequi, sneui, sltui, sgtui, sleui, sgeui
Load	lb, lbu, lh, lhu, lw, lf, movi2fp
Store	sb, sh, sw, sf, movfp2i

Potoki w tej architekturze można modelować w następujący sposób:



Rys. 2 Architektura procesora DLX

Każdemu potokowi odpowiada instrukcja w pamięci. Nie każda instrukcja może być zlokalizowana w dowolnym miejscu z powodu różnych jednostek funkcjonalnych zob. Tabela 1, np. instrukcje typu Multiply wykonywane mogą być tylko w 5 potoku.

Przy projektowaniu i optymalizacji oprogramowania należy pamiętać o zwłoce poszczególnych jednostek funkcjonalnych.

2 Zadania do wykonania

2.1 Konfiguracja wstępna

Z1. Uruchom symulator, wgraj przykładowy program *exampl.vdlx*, przekompiluj go i uruchom w systemie krokowym.

Z2. Spróbuj zmodyfikować dowolnie program, żeby zaobserwować zachowanie kompilatora.

Z3. Zaimplementuj program z poprzedniego ćwiczenia (ewentualnie w razie uzasadnionych powodów program mnożenie macierzy 4x4) dla bieżącej architektury VLIW. Wykorzystaj informacje z wykładu dotyczące kolejnych kroków tworzenia takiego oprogramowania.
Z4. Uruchom opcję Language/Cestina ☺

2.3 Zaawansowane

Z4. Zoptymalizuj wykonanie programu, głownie chodzi o technikę *instruction reordering*