Witold Komorowski*

Ewolucja ISA – wierzchołek góry lodowej

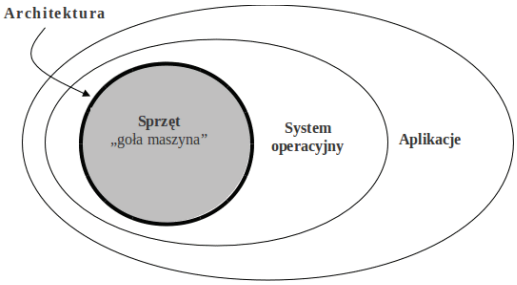
1. *Arché*, czyli zasada

W latach sześćdziesiątych ubiegłego wieku twórcy rodziny komputerów IBM S/360 wprowadzili do opisu urządzeń technicznych, w szczególności projektowanych przez siebie maszyn cyfrowych, nowe znaczenie nobliwego słowa „architektura". Architektura miała teraz oznaczać opis „zachowania się" maszyny z punktu widzenia programisty i odpowiadać na pytanie „co użytkownik musi o maszynie wiedzieć?"[1]. Ponieważ użytkownik-programista może korzystać ze sprzętu jedynie tworząc program w języku danej maszyny, więc podstawowym atrybutem architektury komputera jest jego lista rozkazów. Definiując zbiór rozkazów, trzeba m.in. określić dostępne zasoby (pamięć, rejestry), typy danych i sposób ich kodowania, kodowanie samych rozkazów, sposób adresowania, reakcje na sytuacje wyjątkowe (system przerwań) itp. Wszystkie wymienione elementy są konieczne zwłaszcza przy pisaniu programów w asemblerze, czyli języku najniższego poziomu – odwzorowującym dokładnie język maszynowy komputera. W tym sensie architektura danego komputera określa jego model programowy i daje interfejs (rys.1) między sprzętem (*hardware*) i oprogramowaniem (*software*).

Efektowność nowego terminu stwarzała pokusę jego nadużywania i doprowadziła do pewnego rozmycia znaczeniowego – architekturą nazywa się obecnie ogólną budowę systemu, czasem konfigurację sprzętu; powstało nawet pojęcie mikroarchitektury rozumianej jako struktura logiczna bloków funkcjonalnych. W tym zamieszaniu pojęciowym – dla zaznaczenia, że chodzi o pierwotny sens – zaczęto używać nieco rozbudowanej i tchnącej żargonem nazwy „architektura listy rozkazów" – w skrócie ISA (*Instruction Set Architecture*). Przy okazji zdefiniowano inne aspekty opisu systemu – jego organizację (jak działa?) i realizację (z czego jest zbudowana?).

* doc. dr inż. Witold Komorowski, Dolnośląska Wyższa Szkoła Przedsiębiorczości i Techniki w Polkowicach.

[1] Autorem terminu *system architecture* jest Frederick P. Brooks (*Planning a Computer System: Project Stretch*, red. W. Buchholz, 1962). F.P. Brooks był też później jednym z projektantów IBM S/360. Twórcy tego nowego znaczenia słowa „architektura" wywodzili je z pierwotnego sensu greckiego *arché*, czyli początek, zasada.



Rys. 1. Architektura jako interfejs między sprzętem a oprogramowaniem

Najbardziej widoczną cechą sprzętu jest realizacja – stała się nawet podstawą podziału komputerów na generacje: zerowa – maszyny elektromechaniczne, pierwsza – lampowe, druga – tranzystorowe i trzecia – zbudowane z układów scalonych (symbol S/360 rozwijano jako *System Three-Sixty* – system trzeciej generacji na lata 60. XX w.). *Nota bene* próby kontynuowania tego podziału okazały się nieudane (np. japoński program rządowy Fifth Generation Computers z lat 80. XX w.), gdyż zmiany w technologii elektronowej są bardzo dynamiczne i następują właściwie w sposób ciągły, ale nie ma tak spektakularnych przełomów, jak to było np. przy przejściu z lamp próżniowych na półprzewodniki.

Organizacja (czasem uściślana jako organizacja logiczna) określa strukturę – na różnych poziomach szczegółowości – i opisuje przepływy informacji w tej strukturze. Organizacja jest niezależna od architektury w tym sensie, że identyczna architektura może być implementowana w różnych strukturach, czego przykładem są liczne „klony” popularnych komputerów, zarówno w przeszłości (np. Jednolity System EMC Riad, Siemens 4004, Spectra/70 – wszystkie kompatybilne z modelem IBM S/360), jak i obecnie (rywalizacja firm Intel i AMD w dziedzinie procesorów o architekturze x86). Organizacja jest też niezależna od realizacji, czego jaskrawym przykładem jest przejście w roku 1959 z wersji lampowej dość popularnego wówczas komputera typu IBM 709 do sześciokrotnie szybszego modelu IBM 7090 opartego na tranzystorach.

Przy szybkim rozwoju technologii i stosunkowo częstych zmianach organizacji logicznej, architektura pozostaje bardzo trwałą, konserwatywną cechą komputerów. Wynika to z tego, że przenośność oprogramowania (w tym również niesłychanie pracochłonnych i kosztownych systemów operacyjnych) jest możliwa jedynie między maszynami kompatybilnymi, czyli mającymi tę samą architekturę. Ta zasada jest widoczna w produktach, które odniosły sukces komercyjny, mają miliony użytkowników i obrosły tysiącami aplikacji. Rozwój następuje tu przez zmiany w technologii i organizacji przezwyciężające bariery stwarzane przez architekturę często zdefiniowaną kilkadziesiąt lat wcześniej. Najbardziej oczywistym ograniczeniem jest wielkość adresu; np. wspomniany System/360 zaprojektowano dla adresów 24-bitowych, a które dawały dostęp do olbrzymiego, jak się wówczas wydawało, obszaru 16 MB pamięci, ale już

w latach 70. XX w. była to istotna bariera rozwoju, która do końca lat 90. była omijana w dość skomplikowany sposób w systemach 3033 i S/900. Podobna prawidłowość miała miejsce w 16-bitowych procesorach typu Intel 8088/86, które – trochę przypadkowo – stały się rdzeniem komputerów osobistych IBM PC w 1981 roku. Można w nich było adresować zaledwie 1 MB pamięci i przy takim ograniczeniu opracowano system operacyjny MS DOS. Dopiero w modelu Intel 386 (rok 1986) wprowadzono niejako dwie architektury – starą, 16-bitową i nową (tzw. tryb chroniony), 32-bitową dającą 4 GB przestrzeni adresowej; niemniej do dziś w procesorach Intel stosowanych w PC jest dostępna (w tzw. trybie rzeczywistym) architektura x86 zdefiniowana w 1978 roku.

Trzy wymienione aspekty opisu systemu są także trzema obszarami projektowania. Jakkolwiek można je traktować rozłącznie, to wzajemnie na siebie wpływają, a nowe technologie stwarzają możliwości rozwiązań strukturalnych poprzednio nieefektywnych lub zbyt kosztownych (np. pamięci *cache*, procesory wielordzeniowe). Z drugiej strony koncepcja architektury zależy od rozwiązań strukturalnych (np. sterowanie mikroprogramowe umożliwiło tworzenie złożonych rozkazów typu CISC, a 20 lat później potrzeba pełnego wykorzystania potokowej organizacji przetwarzania dała asumpt do zredukowania listy rozkazów i ich uproszczenia w architekturze RISC).

W miarę rozwoju techniki komputerowej pierwotne kryteria oceny maszyn, takie jak szybkość procesora i wielkość pamięci, ustąpiły całościowemu kryterium wydajności (*performance*) mierzonej szybkością wykonania programów testowych (*benchmarks*). Czas wykonania programu (t) jest iloczynem liczby wykonanych rozkazów (r), liczby cykli zegara potrzebnych do wykonania jednego rozkazu (c) i czasu cyklu zegara ($\frac{1}{f}$), gdzie f jest częstotliwością taktowania:

$$t = rc\frac{1}{f}$$

Przy takiej metodologii szacowania wydajności uwzględniane są wszystkie czynniki natury sprzętowej, ale także programowe (sprawność kompilatora), co ilustruje rys. 2. Dzięki temu kryterium jest miarodajne dla użytkownika, któremu zależy nie tyle na nowoczesności rozwiązań technicznych czy elegancji architektury, ale na szybkości wykonywania jego programów.

Wszystkie te czynniki znajdują odzwierciedlenie w liście rozkazów, która z punktu widzenia programisty jest wizytówką każdego komputera. Analizując repertuar rozkazów różnych komputerów – od pierwszych maszyn z lat 40. ubiegłego wieku aż do dziś – często trudno znaleźć wytłumaczenie takiego, a nie innego ich doboru. Dopiero uwzględnienie wszystkich czynników związanych z dostępną w danym okresie technologią, kosztami realizacji oraz przewidywanymi zastosowaniami wyjaśnia, skąd się wzięły pewne konstrukcje i dlaczego brakowało innych, wydawałoby się wygodniejszych.

Prezentowane dalej przykłady maszyn reprezentatywnych dla rozwoju techniki cyfrowej w jej przełomowych okresach dają pogląd na te ukryte zależności,

	r	c	f
Realizacja (technologia)			
Organizacja (struktura)			
Architektura (lista rozkazów)			
Oprogramowanie (kompilator)			

Rys. 2. Czynniki wpływające na wydajność komputera

a przy okazji pokazują zdumiewającą trwałość pewnych koncepcji pochodzących jeszcze z pionierskiego okresu komputerów sprzed 70 lat.

2. Początki

Pierwsze komputery były budowane jako maszyny liczące, stanowiąc elektroniczną kontynuację rozmaitych arytmometrów budowanych od XVII w., a następnie XX-wiecznych maszyn mechanicznych i elektromechanicznych, zwanych czasem mechanograficznymi (*mécanographie*) [2]. Komputer z definicji tym różni się od swych poprzedników, że posiada pamięć, w której zapisywany jest program jego działania – w tym sensie słynny ENIAC nie był komputerem w dzisiejszym tego słowa znaczeniu. Pierwszy komputer z zapamiętywanym programem wg koncepcji przedstawionej w tzw. raporcie von Neumanna[2] został zbudowany w Europie, na Uniwersytecie Cambridge i uruchomiony w maju 1949 roku (proponowany w raporcie EDVAC został uruchomiony w USA dopiero w 1952 r.). Tym pierwszym komputerem był EDSAC (*Electronic Delay Storage Automatic Calculator*), dzieło M.V. Wilkes'a, który zastosował pamięć na rtęciowych liniach opóźniających, sygnalizując ten fakt w nazwie maszyny (32 rury przechowujące w sposób dynamiczny łącznie 1024 słowa 18-bitowe).

Lista rozkazów EDSAC'a (tab. 1) ma zaledwie 18 pozycji, wśród których można wyróżnić wszystkie grupy rozkazów obecnie spotykane, choć widać ukierunkowanie na wykonywanie obliczeń i chęć jak najmniejszej komplikacji sprzętu. Te podstawowe rodzaje rozkazów to:

- rozkazy arytmetyczne – dodawanie, odejmowanie i mnożenie (A, S, V, N) oraz zaokrąglanie (Y); dzielenie trzeba było programować,
- jeden rozkaz logiczny – AND (C),
- rozkazy przesunięcia w lewo i w prawo (L, R),
- rozkazy zapamiętywania akumulatora w pamięci (T, U) oraz ładowania rejestru mnożnej (H),
- skoki warunkowe (E, G),

[2] First Draft of a Report on the EDVAC z 1945 r. http://qss.stanford.edu/~godfrey/vonNeumann/vnedvac.pdf

Tab. 1. Lista rozkazów komputera EDSAC

Rozkaz	Funkcja
A n	Dodaj liczbę spod adresu n do akumulatora
S n	Odejmij liczbę spod adresu n od akumulatora
H n	Skopiuj liczbę spod adresu n do rejestru mnożnej
V n	Pomnóż liczbę spod adresu n przez liczbę z rejestru mnożnej i dodaj iloczyn do akumulatora
N n	Pomnóż liczbę spod adresu n przez liczbę z rejestru mnożnej i odejmij iloczyn od akumulatora
T n	Prześlij liczbę z akumulatora pod adres n i wyzeruj akumulator
U n	Prześlij liczbę z akumulatora pod adres n i nie zeruj akumulatora
C n	Pomnóż logicznie (AND) liczbę spod adresu n przez liczbę z rejestru mnożnej i dodaj wynik do akumulatora
R 2^{k-2}	Przesuń liczbę w akumulatorze o k pozycji w prawo
L 2^{k-2}	Przesuń liczbę w akumulatorze o k pozycji w lewo
E n	Jeżeli znak akumulatora jest dodatni, skocz do adresu n; jeśli nie – wykonuj kolejny rozkaz
G n	Jeżeli znak akumulatora jest ujemny, skocz do adresu n; jeśli nie – wykonuj kolejny rozkaz
I n	Czytaj następny znak z taśmy papierowej i zapamiętaj go jako 5 mniej znaczących bitów w pamięci pod adresem n
O n	Drukuj znak reprezentowany przez 5 bardziej znaczących bitów w pamięci pod adresem n
F n	Czytaj ostatni wyprowadzony znak do weryfikacji
X	Nic nie rób
Y	Zaokrąglij liczbę w akumulatorze do 34 bitów
Z	Zatrzymaj maszynę i daj sygnał dzwonkiem

• rozkazy wejścia–wyjścia (I, O, F).

Był nawet rozkaz NOP („nic nie rób") i nieco zabawnie wyglądający rozkaz Z. M.V. Wilkes był nie tylko twórcą EDSAC'a, ale też autorem pierwszego asemblera; jednoliterowe skróty nazw rozkazów wypisane w tab. 1 stanowiły w 5-bitowym kodzie telegraficznym właśnie ich symboliczne kody operacyjne stosowane w tym asemblerze.

Ze względu na to, że była to maszyna jednoadresowa z jednym rejestrem akumulatora i dodatkowym rejestrem mnożnej, więc wszystkie działania miały jeden argument domyślny, a drugi był pobierany z pamięci. Zwraca uwagę ascetyczność tej listy wynikająca z chęci uproszczenia struktury elektronicznej bez względu na ewentualne komplikacje programowania – nie ma np. rozkazu ładowania akumulatora, gdyż można to uzyskać przez dodawanie do pustego (wyzerowanego) rejestru. Brak rejestrów indeksowych powodował, że np. przetwarzanie bloku danych wymagało zmiany adresu w odpowiednim rozkazie,

zmiany dokonywanej w czasie wykonywania programu (program samomodyfikujący). Taka możliwość traktowania słowa rozkazowego jako argumentu operacji wynika z definicji komputera, ale obecnie byłaby traktowana jako błąd w sztuce, jeżeli w ogóle mechanizmy ochrony dopuściłyby taką sytuację.

Rozkazy przesunięć dają kolejny przykład podporządkowania projektu logicznego wygodzie konstrukcyjnej: liczba pozycji, o którą należy przesunąć zawartość akumulatora, nie jest określona przez wartość pola adresowego, ale przez położenie skrajnej „1" w słowie rozkazowym. Np. rozkaz „R 8" powoduje przesunięcie o 5 pozycji, a nie o 8 jakby sugerował zapis ($8 = 2^{5-2}$ i ma postać dwójkową 01000).

Bardzo ubogi jest repertuar skoków – brak podstawowego obecnie narzędzia organizacji podprogramów (procedur), czyli rozkazu typu *call*. Podprogramy zapamiętywały adres powrotu jako część jednego ze swych rozkazów „składanego" w czasie wywołania z programu głównego. W ogóle nie ma skoku bezwarunkowego, a dwa skoki warunkowe badają tylko jeden warunek – znak wyniku w akumulatorze; bezwarunkowa zmiana sterowania była realizowana przez kolejne użycie obu skoków warunkowych.

Ówczesny stan techniki oddają rozkazy wejścia–wyjścia. Ponieważ jedynymi urządzeniami zewnętrznymi był dalekopis (wyjście) i czytnik 5-kanałowej taśmy perforowanej (wejście), więc wpisanie słowa polegało na skompletowaniu go w akumulatorze (powtarzana para rozkazów „czytaj znak" – „przesuń w lewo o 5 pozycji") i przesłanie całości do pamięci. Dotyczyło to zarówno danych, jak i kodu programu; zadaniem programisty było przygotowanie sekwencji 5-bitowych znaków odpowiadającej kodowanemu rozkazowi.

Przykłady oryginalnych programów, dokładny opis maszyny i symulator „EDSAC" dla PC zbudowany kilka lat temu na Uniwersytecie Warwick można znaleźć w Internecie [1].

Drugim przykładem maszyn pierwszej generacji jest „Odra 1002", pierwszy działający komputer z Wrocławskich Zakładów Elektronicznych Elwro uruchomiony w 1962 r. [6]. Podstawowym czynnikiem wpływającym na architekturę była tu realizacja pamięci operacyjnej jako pamięci bębnowej dającej cykliczny dostęp do adresów. Dostęp dowolny miały dopiero pamięci ferrytowe (*random access memory*) i wówczas powstał skrót RAM, używany dziś jako synonim pamięci operacyjnej. Stosowanie pamięci bębnowej prowadziło do koncepcji maszyn niesekwencyjnych, wykonujących rozkazy w innej kolejności niż wynikająca z kolejności ich adresów. W maszynie niesekwencyjnej rozkaz zawiera oprócz adresów argumentu również adres następnego rozkazu; np. pierwsza von neumannowska maszyna EDVAC była (3+1)-adresowa, maszyny ODRA były „jeden plus jeden adresowe". Dodatkowy adres umożliwiał optymalne rozmieszczenie rozkazów w pamięci: jeżeli jest znany czas wykonania rozkazu, to rozkaz następny należy umieścić pod adresem, który będzie dostępny do odczytu w momencie zakończenia bieżącego rozkazu.

Szeregowy dostęp do pamięci (bit po bicie) narzucił szeregowy sposób przetwarzania informacji. Cykl zegara wyznaczał czas przesłania (i przetworzenia) jednego bitu; czas ten określano jako *chwilę maszynową*, wielokrotność *chwili* wystarczającą na przesłanie całego słowa była nazwana *krokiem*. Dla zachowania synchronizacji z pamięcią impulsy zegarowe były generowane ze specjalnej ścieżki na bębnie, dzięki czemu niewielkie wahania obrotów nie powodowały błędów.

Mimo iż zastosowano bardzo prosty arytmometr – pojedynczy układ sumatora 1-bitowego na wejściu przesuwnego rejestru akumulatora(!) – „ODRA 1002" miała zrealizowane układowo operacje mnożenia i dzielenia stałoprzecinkowego, gdyż, podobnie do sobie współczesnych, była „aparatem matematycznym"[3] z listą rozkazów (tab. 2) ukierunkowaną na obliczenia [14].

Umieszczone w tab. 2 symbole rozkazów nie są mnemonikami w dzisiejszym rozumieniu, ponieważ nie można ich było stosować w programach; „Odra 1002" była programowana w języku maszynowym (niesymbolicznym), a jedynym ułatwieniem było stosowanie zapisu ósemkowego zamiast dwójkowego przy kodowaniu rozkazów i adresów. Rozkazy opisane kursywą nie były stosowane w programach. W opisie rozkazów litery A i M oznaczają rejestry 36-bitowe, F – krótki, 12-bitowy rejestr modyfikacji adresu, p – słowo w pamięci wskazane w rozkazie. Podane czasy wykonania dotyczą tylko fazy wykonania; pełny cykl rozkazowy zawierał czas oczekiwania na rozkaz (maksymalnie 63 kroki), czas pobrania rozkazu (1 krok), czas oczekiwania na argument (maksymalnie 63 kroki) i czas wykonania.

Lista rozkazów maszyny „ODRA 1002" jest typowa dla maszyn jednoargumentowych niesekwencyjnych i oprócz rozkazów spotykanych we współczesnych maszynach ma też działania, które obecnie wydają się nieco egzotyczne.

Głównym rejestrem arytmometru jest tu akumulator A, który zawsze zawiera domyślny argument działań 2-argumentowych. Drugi rejestr arytmometru M, podobnie jak w EDSAC, spełnia pomocniczą rolę przy mnożeniu i dzieleniu.

Rozkazy przesłań typu *load* i *store* zapewniają komunikację tych rejestrów z pamięcią (doA, doM, zA, zM). Do akumulatora można ponadto wpisywać wartość ujemną wskazanego argumentu (odA) i zero (ZrA). Ostatni rozkaz z grupy przesłań ładuje parametr *n* rozkazu do krótkiego rejestru modyfikacji F.

Rozkazy przesunięć (Ap, Cp, Al) dotyczą zawsze zawartości akumulatora, przy czym liczba pozycji jest podawana jako parametr.

Rozkazy arytmetyczne działające na argumentach w kodzie uzupełnieniowym realizują cztery podstawowe działania (Do, Od, Mn, Dz) oraz znajdowanie wartości bezwzględnej (Bw) i liczby przeciwnej (uzA). Mnożenie i dzielenie jest wykonywane układowo, co zajmuje 35 kroków i w działaniach tych bierze udział rejestr M, przechowując mniej znaczącą część iloczynu lub resztę z dzielenia. Ze względu na przyjętą „skalę zerową" (przecinek jest umieszczony po najbardziej

[3] Ten oryginalny termin występował w nazwie Zakładu Aparatów Matematycznych PAN – kolebki polskiej informatyki w latach 1956–1962.

Tab. 2. „ODRA 1002" – Lista rozkazów

Kod operacyjny (ósemkowo)	Symbol	Nazwa	Liczba kroków w fazie wykonania
0		*Nic nie rób*	*1*
1	Sks	Skok ze śladem	1
2		*Nic nie rób*	*1*
3		*Nic nie rób*	*1*
4	Mn	Mnożenie A*p	35
5	Dz	Dzielenie A/p	max 37
6	doM	Ładowanie M	1
7	zM	Zapamiętanie M	1
10	Kon	Koniunkcja A&p	1
11		*Zerowanie A*	*1*
12	Rs	Różnica symetryczna A÷p	1
13	zA	Zapamiętanie A	1
14	doA	Ładowanie A	1
15	odA	Ładowanie ujemne A	1
16	Do	Dodawanie A+p	1
17	Od	Odejmowanie A-p	1
20	We	Wejście	17
21	Wy	Wyjście	129
22	SkW	Skok przy warunku W	1
23	SkN	Skok przy warunku N	1
24	Bw	Bezwzględna wartość |A|	1
25	uzA	Uzupełnienie -A	1
26	Stop	Stop z pobraniem	1
27		*Nic nie rób*	1
30	Ap	Arytmetyczne przesunięcie w prawo o n pozycji	n
31	Cp	Cykliczne przesunięcie w prawo o n pozycji	n
32	Al	Arytmetyczne przesunięcie w lewo o n pozycji	n
33	Okr	Zaokrąglanie	1
34	ZrA	Zerowanie A	1
35		*Zerowanie A*	*1*
36	doF	Przesłanie do F	1
37	Nic	*Nic nie rób*	1

znaczącym bicie liczby) sensowna jest operacja zaokrąglania (Okr) wyniku ułamkowego.

Są tylko dwa rozkazy operacji logicznych – koniunkcja (Kon) i różnica symetryczna (Rs) – działających na odpowiadających sobie bitach akumulatora i wybranego słowa w pamięci.

Ze względu na niesekwencyjne wykonywanie rozkazów specyficzna jest realizacja skoków. Ponieważ każdy rozkaz zawiera adres następnego rozkazu, nie ma potrzeby stosowania skoku bezwarunkowego a dwa skoki warunkowe (SkN i SkW) badają stan znaczników N i W ustawianych po każdej operacji (za wyjątkiem przesłań i samych skoków).

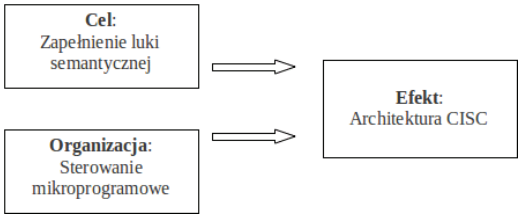
Oryginalną konstrukcję ma wywołanie podprogramu, które odbywa się rozkazem „skok ze śladem" (Sks). Rozkaz ten wpisuje pod wskazanym adresem *n* kod rozkazu „nic nie rób" z adresem następnego rozkazu takim, jaki był w Sks, a następnie przechodzi do rozkazu z komórki *n+2* , gdzie powinien znajdować się pierwszy rozkaz podprogramu. Wybór *n+2* jako adresu kolejnego rozkazu był podyktowany chęcią optymalizacji czasu wykonania: wykonanie rozkazu Sks, czyli zapis do komórki *n*, zajmuje 1 krok, po którym gotowa do odczytu (w tym przypadku celem pobrania rozkazu) jest komórka *n+2*. Powrót do programu głównego następuje przez skok do adresu *n*, symulując w ten sposób adresację pośrednią; podprogramy muszą więc być konstruowane tak, by ich pierwszy adres zawierał „przechowalnię" adresu powrotnego.

Rozkazy wejścia-wyjścia (We i Wy), tak jak w EDSAC, są specjalizowane – dotyczą dwóch konkretnych urządzeń zewnętrznych przewidzianych do współpracy z „Odrą 1002": czytnika i perforatora taśmy papierowej. Dodatkowym urządzeniem był dalekopis spełniający w istocie rolę dekodera z kodu 5-bitowego na postać alfanumeryczną i odwrotnie. Czytnik dostarczał 5-bitowy kod znaku na pozycje <35:31> akumulatora z maksymalną szybkością 300 znaków/s, perforator wyprowadzał na taśmę kod z tych samych pozycji akumulatora z szybkością 30 znaków/s. Z tych szybkości wynikał czas przeznaczony na każdy z rozkazów wejścia–wyjścia, odpowiednio 17 i 129 kroków.

Oba opisane przykłady pokazują, w jak dużym stopniu funkcjonalność nowego sprzętu była zależna od otoczenia technologicznego, gdzie przy projektowaniu maszyn starano się korzystać z istniejących rozwiązań w dziedzinie elektroniki, łączności czy mechaniki precyzyjnej, a wygoda użytkowników była na dalszym planie. To ostatnie było usprawiedliwione faktem, że bezpośrednimi użytkownikami byli stosunkowo nieliczni operatorzy-programiści znający doskonale budowę obsługiwanych urządzeń. Sytuacja zmieniła się na etapie komercjalizacji komputerów, kiedy nastąpiło wyraźne rozdzielenie dziedzin *hardware* i *software* i wkrótce okazało się, że nakłady na oprogramowanie przewyższają koszty sprzętu.

3. *Embarras de richese*

Po wprowadzeniu języków symbolicznych (asemblerów), które stały się głównym narzędziem programowania systemowego (systemy operacyjne, kompilatory), nowe listy rozkazów wyposażano w coraz bardziej złożone instrukcje ułatwiające programowanie. Celem stało się zniwelowanie luki semantycznej między asemblerami, a więc i językami maszynowymi, a językami algorytmicznymi wysokiego poziomu. W komputerach lat siedemdziesiątych, określanych później jako CISC (*Complex Instruction Set Computers*), ta tendencja osiągnęła swe apogeum.



Rys. 3. Sterowanie mikroprogramowe umożliwiło implementację skomplikowanych rozkazów kosztem wydłużenia czasu ich realizacji

Klasycznym przykładem architektury CISC jest superminikomputer VAX [5, 8], który w swej liście rozkazów miał działania wykonywane w innych maszynach przez wcale niebanalne podprogramy (tab. 3).

Rozkazy VAX są różnej długości (od 2 do 57 bajtów), zależnej od wykonywanej operacji, liczby argumentów, typu tych argumentów i sposobu adresowania. Kod operacyjny zapisany w pierwszym bajcie rozkazu (w niektórych przypadkach w 2 bajtach) określa tylko liczbę specyfikatorów, które wystąpią w rozkazie, natomiast ich wielkość zależy od sposobu adresowania.

Np. rozkaz INCW (inkrementuj słowo) ma jeden specyfikator – adres słowa, rozkaz MOVL (prześlij słowo długie) ma dwa specyfikatory będące adresami źródłowym i docelowym przesłania, natomiast rozkaz MOVC5 (kopiuj łańcuch znaków, wypełniając ewentualne wolne miejsca) ma 5 specyfikatorów: długość i adres łańcucha źródłowego, postać bajtu wypełniacza, długość i adres łańcucha docelowego.

Na ogół każdy specyfikator może być kodowany według dowolnego (sensownego w danym kontekście) sposobu adresacji – niezależnie od sposobu kodowania innych specyfikatorów danego rozkazu.

Imponujący jest wykaz typów danych, na jakich mogą operować rozkazy VAX:

- 5 formatów liczb całkowitych ze znakiem lub bez znaku: bajt (**B**yte) – 8b, słowo (**W**ord) – 16b, słowo długie (**L**ongword) – 32b, tetrasłowo (**Q**uadword) – 64b, oktasłowo (**O**ctaword) – 128b; np. zakres liczb w oktasłowie wynosi dla liczb bezznakowych (0, 2^{128}-1) lub (-2^{127}, 2^{127}-1) dla liczb w kodzie U2,
- 5 formatów liczb zmiennopozycyjnych: F, D, G, H i format krótki:

Tab. 3. Przykłady rozkazów asemblera VAX

Mnemonik generyczny	Funkcja	Parametry rozkazu
ADDP6	Dodawanie liczb dziesiętnych o zadanych długościach.	6: adresy i długości obu argumentów i wyniku
POLY_	Obliczanie (metodą Hornera) wielomianu zadanego stopnia; argument, współczynniki i wynik mogą być w jednym z 4 formatów zmp D, F, G lub H; wynik jest umieszczany w rejestrach R0 – R5.	3: stopień wielomianu, adres argumentu, adres tablicy współczynników
MOVTC	Tłumaczenie dowolnego łańcucha znaków wg zadanego kodu (bajt źródłowy jest indeksem tablicy kodowej) z wypełnianiem ewentualnych wolnych bajtów łańcucha docelowego.	6: adres i długość źródłowego łańcucha bajtów, wypełniacz, adres tablicy kodowej, adres i długość łańcucha docelowego
CRC	Obliczanie pola kontrolnego CRC (kodu resztowego) w dowolnym łańcuchu bitów wg zadanego wielomianu; wielomian i wynik są 32-bitowe.	4: adres i długość łańcucha bitów, adres wielomianu, adres wyniku
INSV	Wstawia początkowy fragment podwójnego słowa (src) o rozmiarze size bitów w miejsce określone przez parametry base (adres początku pola) i pos (nr bitu pola, od którego będzie wstawienie).	4: adres i rozmiar wstawianego słowa (src, size), miejsce wstawienia (base, pos)
ACB_	Skok wyliczany: aktualizuje zmienną index, dodając stały przyrost add, po czym nowa wartość indeksu jest porównywana z wartością graniczną limit i następuje skok na odległość displ.	4: adres zmiennej index, przyrost add, wartość graniczna limit i odległość skoku displ

format **F** – 32b, format **D** – 64b, format **G** – 64b, format **H** – 128b; w formacie H zakres wynosi ($0{,}84*10^{-4932}$, $0{,}59*10^{4932}$) przy dokładności 33 cyfr dziesiętnych,
- liczby dziesiętne (spakowane) **P** – do 16 bajtów (31 cyfr dziesiętnych),
- łańcuch znaków **C** – od 0 do 65535 bajtów,
- pole bitowe **V** – od 1 do 32 bitów,
- łańcuch numeryczny **S** 31-bajtowy,
- lista elementów – co najmniej 2 słowa długie.

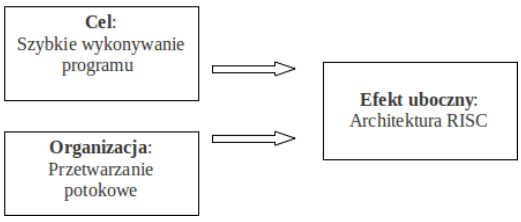
Powyżej czcionką pogrubioną zaznaczono oznaczenia formatu stosowane w mnemonikach asemblera. Np. rozkaz o ogólnej (generycznej) postaci MOV_ src, dst powoduje przesłanie argumentu src na miejsce dst; w wersji MOVB przesyłane są bajty (B), MOVW – słowa (W) itd. Podobnie, mnemonik DIVP oznacza rozkaz dzielenia liczb dziesiętnych spakowanych (P), a mnemonik CVTPS – zamianę liczby dziesiętnej spakowanej (P) na łańcuch numeryczny (S).

Lista rozkazów VAX zawiera ok. 400 pozycji, przy czym rozkazy wieloargumentowe są ortogonalne, tzn. poszczególne argumenty mogą być w nich kodowane wzajemnie niezależnie, w dopuszczalny dla każdego z nich sposób. Lista jest symetryczna względem typów danych, co oznacza, że jeżeli rozkaz ma sens dla pewnego typu danych, to z reguły istnieje.

Konsekwencjami programistycznymi koncepcji CISC jest elegancja (ortogonalność i symetria listy rozkazów), duży repertuar rozkazów, różnorodność sposobów adresowania argumentów (np. 21 sposobów adresowania w VAX), wyrafinowane operacje i typy danych. Z drugiej strony bogactwo rozkazów maszynowych prowadziło do tego, że rozkazy były długie i różnej długości, co komplikowało strukturę i działanie układu sterującego. Co więcej, analizy stosowania rozkazów w tych maszynach pokazywały, że niektóre wyrafinowane instrukcje były jedynie sporadycznie używane przez programistów. Spektakularnym argumentem przeciw nadmiernie rozbudowanej liście rozkazów był pierwszy zrealizowany w technologii mikroprocesorowej MicroVAX. W tym procesorze zaimplementowano w postaci mikrokodu tylko 175 rozkazów i tylko 6 z 14 możliwych typów danych, a reszta była emulowana. Mimo tych ograniczeń, wymuszonych koniecznością uproszczenia układu scalonego, ten podzbiór rozkazów obsługiwał ok. 98% rozkazów wykonywanych przez typowe programy. Te spostrzeżenia były jednym z argumentów za wprowadzeniem architektury RISC.

4. Ryzykowny RISC

Pod koniec lat 70. XX w. nastąpił odwrót od osobnego rozpatrywania architektury i jej implementacji; celem stało się uzyskanie jak największej wydajności systemu, a więc i szybkości działania poprzez uwzględnienie trzech czynników (por. rys. 2): architektury, organizacji, jak również oprogramowania (kompilatorów) [7]. Efektem takich analiz było pojawienie się na początku następnej dekady kilku procesorów eksperymentalnych określanych – od pionierskiej konstrukcji z Uniwersytetu Berkeley – jako RISC (*Reduced Instruction -Set Computer*).



Rys. 4. Uproszczona lista rozkazów w architekturze RISC jest efektem ubocznym dążenia do uzyskania możliwie największej wydajności przetwarzania

W projekcie procesora Berkeley RISC założono realizację potokową strumienia rozkazów i podporządkowanie architektury możliwie prostej strukturze sterowania. Z tego względu przyjęto stałą długość słowa rozkazowego; rozkaz zawsze ma 32 bity i występuje tylko w 3 formatach:

- kod operacji (7b) i 3 adresy rejestrów (3 x 5b),
- kod operacji (7b), 2 adresy rejestrów (2 x 5b) i stała w kodzie U2 (13b),
- kod operacji (7b), adres jednego rejestru (5b) i adres względny w kodzie U2 (19b).

Dla sprawnej realizacji potokowego wykonywania rozkazów działania wykonuje się wyłącznie na zawartości rejestrów (są 32 rejestry 32-bitowe), a jedynie dwa rozkazy komunikują się z pamięcią: ładowanie rejestru z pamięci i zapisanie rejestru w pamięci. Stało się to konstytutywną cechą koncepcji RISC i z tego powodu czasem jest ona określana jako architektura „*load – store*".

Najbardziej oryginalnym pomysłem jest realizacja rozkazów sterujących przebiegiem programu, czyli wszelkimi skokami (*jmp, call, ret*). Rozkazy te są wykonywane jako „skoki opóźnione", czyli po skoku zawsze jest wykonywany, pobrany już, kolejny rozkaz programu, co pozwala na uniknięcie straty czasu na opróżnienie potoku. Sprawą programisty (lub kompilatora) jest sensowne wykorzystanie rozkazu następującego po skoku, a przed efektywnym przeniesieniem sterowania; w najgorszym razie trzeba tam umieścić rozkaz „nic nie rób".

Cała lista zaledwie 31 rozkazów jest przedstawiona w tab. 4, z zachowaniem większości oryginalnych oznaczeń autorów [12]: S2 – rejestr lub 13-bitowa stała (w słowie rozkazowym), Y – 19-bitowy adres względny (w słowie rozkazowym), Rd, Rs, Rx, Rm – 32-bitowy rejestr, M[adr] – słowo z pamięci wskazane 32-bitowym adresem adr, PC – licznik rozkazów, C – przeniesienie, CWP – wskaźnik okna rejestrów, PSW – słowo stanu procesora, cond – kod warunku (14 możliwych warunków skoku).

Klasyczne architektury RISC przyjmują bardzo uproszczoną listę rozkazów, aby efektywnie wykorzystać nowoczesne rozwiązania organizacyjne – pamięci buforowe (*cache*) i wielostopniowe sterowanie potokowe. Ten prymitywizm operacji na poziomie języka maszynowego nie jest widoczny dla programisty piszącego programy w języku wyższego poziomu, gdyż przesłania go dobry kompilator. Zakłada się, że programowanie w asemblerze jest potrzebne sporadycznie i dotyczy bardzo małej grupy specjalistów.

Obecnie ascetyczne założenia dotyczące liczności rozkazów nie są spełnione i nowe procesory RISC dysponują np. bardzo rozbudowanymi działaniami zmiennopozycyjnymi czy operacjami na upakowanych argumentach potrzebnymi do przetwarzania danych na użytek multimediów. Niemal wszystkie produkowane teraz procesory są typu RISC z jednym, za to bardzo istotnym wyjątkiem – dominującą na rynku komputerów osobistych architekturą x86, przy czym od kilku lat procesory Intel, zachowując zewnętrzną architekturę IA-32, zawierają jądro o innej architekturze – wewnętrzny RISC.

Tab. 4. Rozkazy RISC I

Mnemonik	Działanie	Funkcja
ADD	Rd → Rs + S2	Dodawanie
ADC	Rd → Rs + S2 + C	Dodawanie z przeniesieniem
SUB	Rd → Rs – S2	Odejmowanie
SUBC	Rd → Rs – S2 – C	Odejmowanie z przeniesieniem
SUBR	Rd → S2 – Rs	Odejmowanie odwrotne
SUBCR	Rd → S2 – Rs – C	Odejmowanie odwrotne z przeniesieniem
AND	Rd → Rs & S2	Logiczne AND
OR	Rd → Rs v S2	Logiczne OR
XOR	Rd → Rs ÷ S2	Logiczne XOR
SLL	Rd → Rs. Przes. w lewo o S2	Przesunięcie naturalne w lewo
SRL	Rd → Rs. Przes. w prawo o S2	Przesunięcie naturalne w prawo
SRA	Rd → Rs. Przes. arytm.w prawo o S2	Arytmetyczne przesunięcie w prawo
LDL	Rd → M[Rx + S2]	Ładowanie długie (32b)
LDSU	Rd → M[Rx + S2]	Ładowanie krótkie (16b)
LDSS	Rd → M[Rx + S2]	Ładowanie krótkie (16b) z powieleniem znaku
LDBU	Rd → M[Rx + S2]	Ładowanie bajta
LDBS	Rd → M[Rx + S2]	Ładowanie bajta z powieleniem znaku
STL	M[Rx + S2] → Rm	Zapamiętanie długie (32b)
STS	M[Rx + S2] → Rm	Zapamiętanie krótkie (16b)
STB	M[Rx + S2] → Rm	Zapamiętanie bajta
JMP	(cond) ⇒ PC → Rx + S2	Skok warunkowy
JMPR	(cond) ⇒ PC → PC + Y	Skok warunkowy względny
CALL	CWP –; Rd → PC next PC → Rx + S2	Wywołanie procedury i zmiana okna
CALLR	CWP –; Rd → pc next PC → Rx + Y	Wywołanie procedury względne i zmiana okna
RET	PC → Rx + S2 next CWP ++	Powrót z procedury i zmiana okna rejestrów
RETINT	PC → Rx + S2 next CWP ++	Powrót z procedury, zmiana okna rejestrów i otwarcie przerwań
CALLINT	CWP –; Rd → PC	Zapisanie licznika rozkazów i zmiana okna
LDHI	Rd<31:13> → Y; Rd<12:0> → 0	Wpisanie stałej (19 bitów b. znacz.)
GTLPC	Rd → PC	Zapisanie licznika rozkazów
GETPSW	Rd → PSW	Zapisanie słowa stanu
PUTPSW	PSW → Rm	Ustawienie słowa stanu

Bezpośrednią adaptacją pionierskiej architektury mikroprocesorów opracowanych na Uniwersytecie w Berkeley – RISC I i RISC II (1982) jest SPARC (*Scalable Processor Architecture*) – definicja otwartej architektury ogłoszona przez Sun Microsystems w 1987 r. Nie jest to nazwa konkretnej implementacji, choć wiele procesorów i stacji roboczych ma ten skrót w swojej nazwie. W ciągu minionych lat sprawdziła się zarówno idea skalowalności, jak i otwartości. O skalowalności świadczy fakt, że procesory SPARC są stosowane zarówno w notebookach, stacjach roboczych, jak i w superkomputerach (np. SuperServer 6400 wyprodukowany przez Cray Research w 1993 r. był zbudowany z 64 procesorów SPARC). Druga cecha – otwartość architektury, polegająca na ogólnym udostępnieniu specyfikacji, zaowocowała jej realizacją przez kilkunastu producentów. W 1994 r. ogłoszono, po 3 latach prac grupy *SPARC Architecture Committee,* wersję 64-bitową SPARC-V9 będącą podstawą współczesnych implementacji. Wersja V9 jest kompatybilna z 32-bitową wersją V8.

Obecnie procesory Sun SPARC są wyposażane, oprócz standardowej listy rozkazów stałoprzecinkowych, w rozkazy zmiennoprzecinkowe (formaty 32-, 64- i 128-bitowe) oraz w rozkazy VIS (*Visual Instruction Set*) realizujące jednoczesne działania na zestawie bajtów, półsłów i słów – podobnie jak w rozkazach MMX w architekturze procesorów Intel x86.

5. Oszałamiająca równoległość

Zwiększenie wydajności obliczeniowej komputerów, od początku ich powstania, osiągano dwiema drogami: przez postęp technologiczny dający coraz szybsze układy elektroniczne oraz przez zmiany w organizacji logicznej. Istotą tych ostatnich zawsze jest wprowadzenie równoległości (jednoczesności) wykonywania różnych etapów działań.

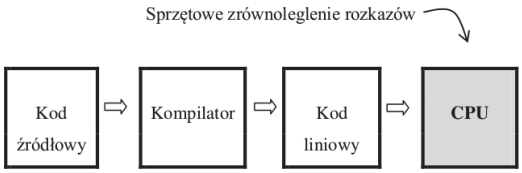
W okresie pionierskim takim skokiem było przejście z przetwarzania szeregowego, bit po bicie na przetwarzanie równoległe całego słowa maszynowego w jednym cyklu zegara. Zwiększenie długości słowa, nawet przy zachowaniu innych cech, skutkowało większą wydajnością; i tak w latach 60. XX w. odniósł sukces komercyjny minikomputer PDP-8, który miał słowo o długości zaledwie 12 bitów, a najpotężniejszy wówczas komputer CDC 6600 przetwarzał słowa 60-bitowe.

Bardziej wyrafinowana metoda podwyższenia wydajności polega nie na przyspieszeniu wykonania poszczególnych działań, a na zmianie sposobu sterowania, tak aby przyspieszyć wykonanie sekwencji rozkazów, czyli programu (*instruction-level parallelism*). Można to uzyskać przez równoczesne wykonywanie kilku rozkazów przez różne jednostki wykonawcze procesora. Ten pomysł jest podstawą przetwarzania potokowego (*pipeline*), w którym poszczególne fazy cyklu rozkazowego mają swoje specjalizowane „stanowiska obsługi", które działają na zasadzie taśmy produkcyjnej. Przykładowo w potoku 5-stopniowym może

się jednocześnie znajdować pięć rozkazów na różnych etapach zaawansowania, a więc w czasie jednego cyklu zostaje zakończonych pięć kolejnych rozkazów, co daje 5-krotne przyspieszenie w stosunku do realizacji sekwencyjnej.

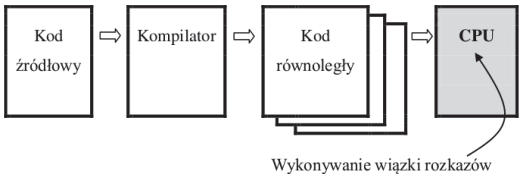
Jeszcze większą równoległość wykonania strumienia rozkazów oferują procesory superskalarne (wielopotokowe), w których z pamięci pobiera się jednocześnie kilka kolejnych rozkazów i są one wydawane (*issue*) do współbieżnego wykonania w różnych jednostkach funkcyjnych, zwykle też potokowych. Wybór rozkazów do emisji i ich przydział do jednostek wykonawczych musi uwzględniać zależności pomiędzy rozkazami wykonującymi się jednocześnie, a operującymi czasem na tych samych danych. W procesorach superskalarnych ten wybór i rozstrzyganie ewentualnych konfliktów odbywa się sprzętowo w bloku dyspozytorskim na podstawie bieżącej analizy pobieranego z pamięci strumienia rozkazów z uwzględnieniem zależności występujących między rozkazami. Przy wielu jednostkach wykonawczych procesora superskalarnego ich pełniejsze wykorzystanie może być uzyskane przez wykonywanie rozkazów w innej kolejności niż kolejność zapisana w programie (*out-of-order*). W tym celu stosuje się pomocnicze rejestry robocze do chwilowego przechowywania argumentów na czas obliczeń współbieżnych. Po zakończeniu działań, w fazie kompletacji wyników (*retirement*) – ich zawartość jest przepisywana do rejestrów programowych.

Zasadę działania procesora superskalarnego obrazuje rys. 5: program źródłowy jest tłumaczony na kod wykonywalny mający postać sekwencji rozkazów ułożonych w logicznej kolejności ich wykonywania, a dopiero wysoce skomplikowany układ dyspozytora w procesorze wybiera kilka rozkazów do równoczesnego wykonania.



Rys. 5. Zasada przetwarzania w procesorze superskalarnym

Inny sposób zrealizowania tej koncepcji oferują tzw. komputery z bardzo długim słowem rozkazowym (VLIW, *Very Long Instruction Word*) – tutaj następuje rozdzielenie funkcji równoległego wykonywania operacji od funkcji przygotowania zestawu tych operacji, czyli wydzielenia ich ze strumienia rozkazów programu sekwencyjnego. Procesor VLIW interpretuje długie słowo rozkazowe będące zastawem kilku zwykłych rozkazów typu RISC, przygotowanym poza procesorem na etapie kompilacji (rys. 6). Dzięki temu unika się skomplikowanej i nie zawsze skutecznej analizy wzajemnych zależności dokonywanej sprzętowo na bieżąco, a w efekcie działanie procesora może być bardziej efektywne.



Rys. 6. Zasada przetwarzania w procesorze VLIW

Obecnie najpełniej zasadę VLIW realizuje architektura IA-64 (*Intel Architecture-64*) [7] występująca pod nazwą EPIC (*Explicitly Parallel Instruction Computing*), wprowadzając jawny podział funkcji między sprzęt (bardzo wydajny procesor odciążony od zadań optymalizacji kodu) a oprogramowanie (skomplikowany kompilator przygotowujący wiązki rozkazów możliwe do bezkonfliktowego wykonania w procesorze). Jest to kontynuacja tendencji, która była podstawą procesorów RISC – uproszczenie organizacji logicznej procesorów (i dzięki temu przyspieszenie ich działania) kosztem nieuniknionej komplikacji oprogramowania.

Technika ta, opracowywana wspólnie przez firmy Intel i Hewlett Packard, zakłada, że procesor wykonuje 128-bitowe wiązki (*bundle*) zawierające po trzy 41-bitowe rozkazy typu RISC. Oprócz rozkazów każda wiązka ma 5-bitowy szablon określający, które jednostki funkcyjne procesora są potrzebne do jej wykonania. Wiązka rozkazów jest tworzona statycznie na etapie kompilacji (*smart compiler*), a nie – dynamicznie, w czasie wykonywania.

Architektura IA-64 przyjmuje podstawowy format 64-bitowy, co oznacza używanie 64-bitowych danych i 64-bitowych adresów, a w konsekwencji – teoretycznie – dostęp do 2^{64} czyli 16 EB (ok. $18*10^{18}$bajtów) przestrzeni adresowej.

Oprócz olbrzymiej pamięci operacyjnej, procesor dysponuje wielką liczbą rejestrów dla argumentów wymaganych przez typowe 3-adresowe rozkazy RISC-owe. Argumenty stałopozycyjne są przechowywane w 64 rejestrach 64-bitowych, zaś argumenty zmiennopozycyjne w 64 rejestrach 82-bitowych. Z każdym rejestrem danych skojarzony jest jednobitowy znacznik określający poprawność przechowywanych danych. W działaniach wspierających zastosowania multimedialne (typu rozkazów MMX znanych z architektury IA-32) rejestry stałopozycyjne są traktowane jako zestawy zawierające 8x8, 4x16 lub 2x32 bitów, a rejestry zmiennopozycyjne – jako zestawy dwóch liczb o tym samym znaku i tym samym 17-bitowym wykładniku oraz 32-bitowej części ułamkowej.

Najnowszą, pokazaną w lutym 2011 r., implementacją architektury IA-64 jest 8-rdzeniowy procesor Itanium Poulson [3] zdolny do jednoczesnej emisji 12 rozkazów. Układ scalony o rozmiarach 18,2 x 29,9 mm zawiera dodatkowo olbrzymią pamięć *cache* o pojemności 54 MB przechowującą najczęściej używane dane. Parametry tego procesora zestawiono w tab. 5 z parametrami pierwszego historycznie mikroprocesora, 4-bitowego Intel 4004 sprzed 40 lat.

Tab. 5. Porównanie nieporównywalnych. Mikroprocesory

	Intel 4004	**Itanium Poulson**
Data prezentacji	1.11.1971	1.02.2011
Długość słowa	4 b	64 b
Wielkość adresu	12 b	64 b
Słowo rozkazowe	8 b i 16 b	128 b[a]
Liczba rejestrów	16 (4 b)[b]	128 (64 b) + 128 (82 b)
Przestrzeń adresowana	4 kB	16 EB[c]
Pamięć cache	–	54 MB
Zegar	740 kHz	1,73 GHz
Technologia	10 μm	32 nm
Powierzchnia płytki	12 mm^2	545,8 mm^2
Liczba tranzystorów	2 300	3 100 000 000

[a] Trzy 41-bitowe rozkazy typu RISC + 5 b szablonu w wiązce VLIW.
[b] Plus 3 rejestry stosu adresów powrotnych i licznik rozkazów.
[c] Używa się tylko 50 b adresu, co daje dostęp do 1024 TB.

6. Komputer elementem komputera

W listopadzie 2010 w 36. edycji TOP500, od lat uznanym rankingu superkomputerów [13], najszybszym okazał się chiński Tianhe-1 A („Droga Mleczna", dosłownie „Niebiańska Rzeka") zainstalowany w Narodowym Centrum Superkomputerowym w Tianjin. Jego wydajność wg oficjalnych testów Linpack dochodzi do 2,57 petaflops, czyli 2,57*10^{15} (milion miliardów) operacji zmiennopozycyjnych na sekundę, a teoretyczna wydajność szczytowa aż 4,701 petaflops. Tianhe-1 A zawiera 14 336 procesorów Intel Xeon X5670 (pracujących z zegarem 2,93 GHz) i 7 168 procesorów graficznych Nvidia Tesla M2050, nie licząc 2 048 dodatkowych procesorów NUDT FT1000. Sumaryczna pojemność pamięci wynosi 262 TB (262*10^{12} bajtów), a pojemność pamięci dyskowej – 2 PB (2*10^{15} bajtów).

Liczba procesorów w Tianhe-1 A jest dziesięciokrotnie większa niż była liczba tranzystorów w pierwszym mikroprocesorze 40 lat temu. Takie porównanie ilustruje tendencję widoczną od ponad dwudziestu lat w projektowaniu najszybszych komputerów zwanych od lat 70. XX w. superkomputerami. Tego określenia użyto pierwszy raz dla maszyny CRAY 1 (*Cray Research*, 1976), choć uznaje się, że superkomputerem był już CDC 6600 (*Control Data Corporation*, 1963), gdyż zdecydowanie górował mocą obliczeniową nad współczesnymi sobie konkurentami. Projektantem obydwu był Seymour Cray (1925–1996), który wprowadzał własne rozwiązania nie tylko architektury, ale również struktury, a nawet elektroniki; o jego geniuszu świadczy fakt, że zespół budujący CDC 6600 liczył zaledwie około 30 osób [10]. Istniejąca dziś firma Cray Inc. jest jednym z głównych producentów superkomputerów, a nazwa Cray stała się na dwie

Tab. 6. Porównanie nieporównywalnych. Superkomputery

	CRAY-1A	**Tianhe-1A**
Rok uruchomienia	1977	2010
Liczba procesorów	1[a]	21505
Pamięć	8 MB[b]	262 TB
Szybkość	133 * 106 FLOPS	2,57 * 1015 FLOPS
Moc zasilania	115 kW	4,04 MW
Koszt	8,86 mln USD	88 mln USD[c]

[a] Procesor wektorowy.
[b] Dokładnie: 220 słów 64 b.
[c] Plus koszty eksploatacji ok. 20 mln USD rocznie.

dekady niemal synonimem superkomputera.

Komputer Cray-1 A, ze względu na wielką szybkość zegara (80 MHz), miał specjalną konstrukcję pozwalającą skrócić przewody łączące – zbudowano go w postaci otwartego z boku walca otoczonego u dołu półokrągłą „kanapą" kryjącą freonowy system chłodzenia, a całość ważyła ok. 5,5 tony. Był to pierwszy komputer, który przekroczył barierę 100 megaflops (100 milionów operacji zmiennopozycyjnych na sekundę).

Komputery, takie jak Cray-1, uzyskiwały wysoką wydajność obliczeniową dzięki unikatowej architekturze i nowatorskim rozwiązaniom organizacji logicznej (np. przetwarzanie wektorowe, potokowość, kilka specjalizowanych jednostek wykonawczych), a także przez stosowanie najszybszych wówczas układów elektronicznych. Były to jednak komputery zbudowane wokół jednego procesora centralnego przetwarzającego jeden wątek programu.

Od lat dziewięćdziesiątych najszybsze komputery są budowane wg innego pomysłu: jako systemy wieloprocesorowe stosujące standardowe procesory używane np. w serwerach czy w stacjach graficznych. Początkowo było to kilka czy kilkanaście procesorów, później kilkaset, a obecnie kilkudziesiąt tysięcy; ten sposób działania określa się jako masywny paralelizm przetwarzania (*Massively Parallel Processing*). W takich maszynach problemem nie jest architektura poszczególnych procesorów, a dostatecznie szybka i bezkolizyjna komunikacja między nimi i – przede wszystkim – taki system programowy, który zapewni rozdział zadań wykorzystujący wszystkie elementy systemu.

W tab. 7 pokazano wszystkie superkomputery, które w ostatnim rankingu TOP 500 (listopad 2010) [13] mają wydajność większą niż 1 PFLOPS (dwa lata wcześniej był tylko jeden taki komputer – IBM Roadrunner, który w czerwcu 2008 jako pierwszy przekroczył granicę petaflopsową)[4]. „Masywność" przetwarzania

[4] Dane te, od czasu złożenia artykułu w redakcji, zdezaktualizowały się i obecnie – wg rankingu Top500 z listopada 2011 – najszybszy jest japoński K Computer, który, zawierając 705 024 rdzenie, jako pierwszy przekroczył granicę 10 PFLOPS. Drugie miejsce zajmuje Tianhe-1 A. Wśród 10

Tab. 7. Siedem wspaniałych. Superkomputery petaflopsowe

Wydajność (TFLOPS)	**Producent**	**Kraj**	**Komputer**	**Rok**[a]	**Procesory**	Liczba **rdzeni**
2566	NUDT	Chiny	Tianhe-1 A NUDT TH MPP	2010	Intel EM64T Nvidia GPU	186 368
1759	Cray Inc.	USA	Jaguar Cray XT5-HE	2009	AMD x86_64	224 162
1271	Dawning	Chiny	Nebulae Dawning TC3600 Blade	2010	Intel EM64T Nvidia Tesla	120 640
1192	NEC/HP	Japonia	TSUBAME 2.0 HP ProLiant SL390 s	2010	Intel EM64T Nvidia GPU	73 278
1054	Cray Inc.	USA	Hopper Cray XE6	2010	AMD x86_64	153 408
1050	Bull SA	Francja	Tera-100 Bull bullx super-node	2010	Intel EM64T	138 368
1042	IBM	USA	Roadrunner BladeCenter QS22/LS21	2009	Power AMD x86_64	122 400

[a] Rok instalacji lub ostatniej modyfikacji.

dobrze ilustruje w tej tabelce ostatnia kolumna (liczba rdzeni), gdyż właśnie rdzeń (*core*) jest najmniejszą jednostką przetwarzającą informację. Współczesne mikroprocesory, nawet już te stosowane w komputerach domowych, są budowane jako wielordzeniowe. Taki procesor zawiera w jednym układzie scalonym dwa lub więcej procesorów (rdzeni) zwykle z własnymi pamięciami *cache* pierwszego poziomu (L1) i współdzieloną pamięcią *cache* L2. Na przykład stosowane w superkomputerach procesory Xeon (Intel) (tab. 7) są 6- lub 8-rdzeniowe, procesory Opteron (AMD) – 6- lub 12-rdzeniowe, procesory PowerXCell mają 9 rdzeni, a procesory graficzne Tesla (Nvidia) zawierają 10 rdzeni, przy czym każdy z nich może przetwarzać jednocześnie 24 strumienie danych.

Kolosalna moc obliczeniowa superkomputerów, oprócz tego, że ma zastosowania wojskowe, bankowo-finansowe, filmowe i inne bezpośrednio użyteczne, umożliwiła też powstanie nowego rodzaju badań naukowych – „naukę obliczeniową" (*computational science*). Jest to postępowanie badawcze uzupełniające trzy tradycyjne – obserwację i opisywanie zjawisk (nauki obserwacyjne), prowadzenie doświadczeń (nauki eksperymentalne) oraz stawianie hipotez prowadzących do sformułowania teorii matematyzujących rzeczywistość (nauki teoretyczne). Datą graniczną jest rok 1977, w którym opublikowano rozwiązanie problemu

superkomputerów o wydajności przekraczającej 1 PFLOPS jest jeden francuski, po 2 z Japonii i Chin oraz 5 z USA.

czterech barw metodą nierealizowalną bez zastosowania komputera [9].

Analiza struktury superkomputera przypomina „zoomowanie" w Google Earth – od satelitarnego obrazu Ziemi do szczegółów znajomego skwerku przed domem. Podobnie można przejść od pulsującej strumieniami bitów sieci dziesiątków tysięcy procesorów, przez skomplikowaną strukturę każdego z nich, poznać ich bloki funkcjonalne, szczegóły potokowości, organizacji asocjacyjnych pamięci buforowych etc., aby dojść do poziomu rejestrów i bramek logicznych, a dalej po przekroczeniu bram elektroniki zanurzyć się w subtelności technologii elektronowej w środowisku opisywanym w nanometrach, gigahertzach i nanosekundach. W tym sensie superkomputery zasługują na swój przedrostek, gdyż kumulują w sobie wszystko, co najlepsze w tej dziedzinie techniki.

Praktyczny rozwój informatyki zapoczątkowany zdarzeniem technicznym: wynalazkiem maszyny cyfrowej w połowie XX w. spowodował zmiany cywilizacyjne i kulturowe w niebywałej skali i, używając świetnego sformułowania z tytułu książki Zbigniewa Polańskiego [11], „wpędził nas do komputerowego raju". Dotyczy to zarówno dzieci bawiących się inteligentnymi gadżetami, jak i zwykłych użytkowników usług sieciowych (bankomaty, systemy rezerwacji, sklepy internetowe) i wszelakiego rodzaju urządzeń cyfrowych (telefony, laptopy, GPS-y, tablety, kamery, dyktafony itp.).

Nasuwa się tu sarkastyczna uwaga Stanisława Lema z 1999 roku o rozwoju informatyki [4]: „...wydaje mi się to jakoś nierealne, że tak szybko i tak daleko doszliśmy. Nie wiadomo zresztą, po co".

Streszczenie

Ewolucja ISA – wierzchołek góry lodowej

*Lista rozkazów stanowiąca główny atrybut architektury każdego komputera zmieniała się zależnie od dostępnej technologii i wymagań stawianych przez użytkowników. W artykule opisano kilka rozwiązań ISA (*Instruction-Set Architecture*) – kluczowych w historii informatyki, wskazując na uwarunkowania istniejące w czasie ich powstawania. Przedstawiono powody zmiany paradygmatu projektowania CISC-RISC w latach osiemdziesiątych. Scharakteryzowano istotę przetwarzania równoległego – od potokowości, przez superskalarność i organizację VLIW aż do przetwarzania masywnie równoległego w obecnych superkomputerach.*

Summary

ISA Evolution – Tip of the Iceberg

Instruction-set architecture is determined by many factors, such as technology and users' demand. The ISA evolution is illustrated on several examples – milestones in computing history: EDSAC, VAX, Berkeley RISC. The early 80' CISC-RISC turning point in architecture paradigm is explained. A short characteristic of parallel processing

is given – starting from pipelining, through superscalar and VLIW processors up to petaflops supercomputers using Massively Parallel Processing *technique.*

Literatura

[1] Campbell-Kelly M., *A tutorial guide to the EDSAC simulator*, Dostępny w Internecie: http://www.dcs.warwick.ac.uk/~edsac/Software/EdsacTG.pdf [dostęp: 25 kwietnia 2011].

[2] Gawrysiak P., *Cyfrowa rewolucja. Rozwój cywilizacji informacyjnej*, Wydawnictwo Naukowe PWN, Warszawa, 2008.

[3] *Itanium Solutions Alliance*, Dostępny w Internecie: http://www.itaniumsolutions.org [dostęp: 23 lutego 2011].

[4] Jankowska K., Mucharski P., *Rozmowy na koniec wieku*, Znak, Kraków, 1999.

[5] Komorowski W., *Instrumenta computatoria. Wybrane architektury komputerów*, Helion, Gliwice, 2000.

[6] Komorowski W., *Pierwsze polskie komputery ODRA*, Kwartalnik Historii Nauki i Techniki, t. 47, 2, 2002, s. 47.

[7] Komorowski W., *Krótki kurs architektury i organizacji komputerów*, Wydawnictwo Mikom, Warszawa, 2004.

[8] Levy H., Eckhouse R., *Computer Programming and Architecture: The VAX*, Digital Press, 1989.

[9] Lubański M., *Próba oceny różnych stanowisk w filozofii matematyki*, [w:] *Matematyczność przyrody*, M. Heller, J. Życiński, (red.) Wydawnictwo Petrus, 2010.

[10] Murray C., *Superkomputer – historia Seymoura Craya*, Amber, 1997.

[11] Polański Z., *Wpędzeni do komputerowego raju. Rzecz o informatyce dla każdego*, Wydawnictwo Politechniki Krakowskiej, Kraków, 2010.

[12] Sequin C., Patterson D., *Design and implementation of RISC I*, [w:] *Proc. Advanced Course on VLSI Architecture*, Bristol, 1982, dostępny w Internecie: http://www.eecs.berkeley.edu/Pubs/TechRpts/1982/CSD-82-106.pdf [dostęp: 22 lutego 2011].

[13] *Top 500 supercomputer sites*, Dostępny w Internecie: http://www.top500.org/lists [dostęp: 26 maja 2011].

[14] Zuber R., *14. Maszyna cyfrowa «Odra 1002». Instrukcja obsługi i programowania*, Raport techniczny, Elwro, 1962, maszynopis powielany.