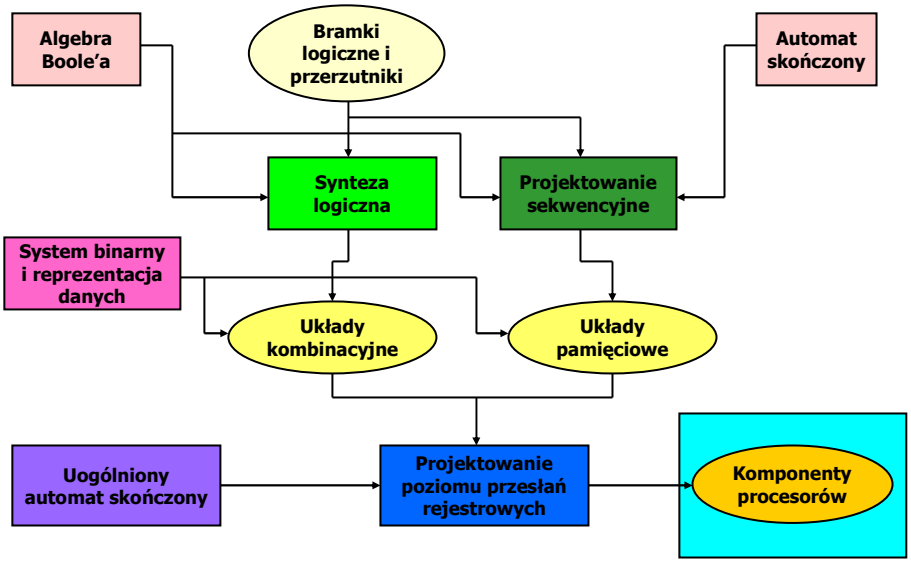
WYKŁAD

Projektowanie mikroprocesorów

Technika cyfrowa

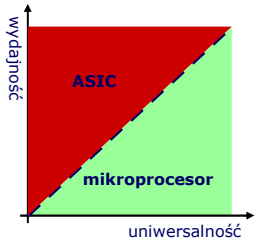
Projektowanie układów cyfrowych - podsumowanie



Rola procesora w systemie komputerowym

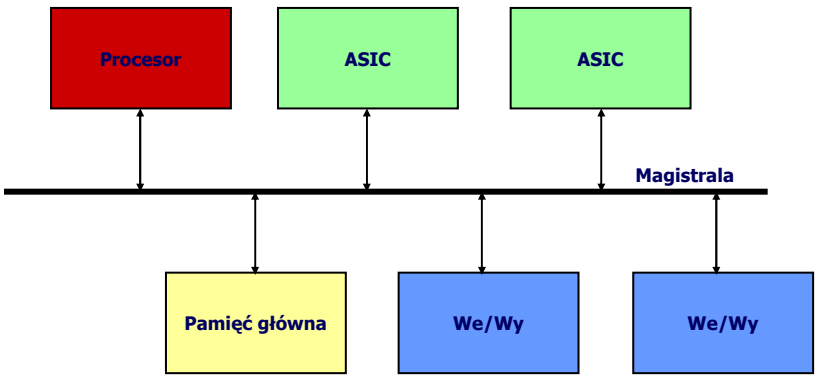
- sterowanie pracą systemu
- nadzorowanie operacji we/wy
- synchronizacja komunikacji pomiędzy elementami systemu
- wykonywanie większości operacji obliczeniowych (z wyłączeniem operacji, które przekraczają jego możliwości -> delegowanie zadań do układów ASIC)

ASIC – wydajne wykonanie pojedynczego zadania lub algorytmu
Mikroprocesor – uniwersalność i programowalność



3

Budowa typowego systemu komputerowego



4

Projektowanie procesorów

Układy ASIC:
opis behawioralny za pomocą algorytmu, programu lub sieci działań.

Mikroprocesor:
zestaw instrukcji

Instrukcja – najmniejsza, niepodzielna jednostka procesu przetwarzania.

Sekwencja instrukcji (program): obliczanie wyrażenia matematycznego lub innego zadania przetwarzania.

Pojedyncze zadanie – specyfikacja za pomocą języka programowania przekształcana za pomocą KOMPILATORA w sekwencję instrukcji.

5

Projektowanie procesorów

Cechy zbioru instrukcji:

- **uniwersalność -> efektywność kompilacji różnych języków programowania**
- **prostota -> wydajność i realizowalność projektu procesora**

Zadania pojedynczych instrukcji:
elementarna arytmetyka
operacje logiczne
operacje przesuwania bitów
realizacja rozgałęzień

6

Zbiór instrukcji

Budowa rozkazu (łańcuch bitów pogrupowanych w różną liczbę pól o różnych rozmiarach):

- kod operacji (*opcode*),
- pole adresu,
- pole typu instrukcji (np. rejestrowe, kopiowania, rozgałęzienia)
- pole trybu (*mode*),
- pole stałej.

Zbiór instrukcji

Wskaźnik liczby pól adresowych w instrukcji -> rozmiar programu i wydajność procesora.

Więcej pól adresowych -> dłuższa instrukcja, krótszy czas wykonania zadania (programu).

Mniej pól adresowych -> krótsze instrukcje, dłuższy program do wykonania.

Dłuższa instrukcja – więcej cykli dostępu do pamięci na jedną instrukcję w celu pobrania samej instrukcji i argumentu.

Zbiór instrukcji - przykład

Operacja: $c=a^2-b^2=(a+b)*(a-b)$

A. Instrukcje zawierające trzy pola adresowe:

1.	Add	X, A, B	(*Mem*[X]<-*Mem*[A]+*Mem*[B])
2.	Sub	C, A, B	(*Mem*[C]<-*Mem*[A]-*Mem*[B])
3.	Mul	C, X, C	(*Mem*[C]<-*Mem*[X]**Mem*[C])

Zakładając współczesne pamięci –> słowo procesora 32 bitowe. Trzy adresy w instrukcji –> trzy słowa adresu.

Trzy cykle dostępu do pamięci w celu pobrania instrukcji i dalsze trzy na pobranie argumentów.

Łącznie w programie 9 cykli dostępu do pamięci programu (pobranie instrukcji) i 9 cykli dostępu do pamięci danych (pobranie argumentów).

Zbiór instrukcji - przykład

Operacja: $c=a^2-b^2=(a+b)*(a-b)$

B. Instrukcje zawierające dwa pola adresowe:

1.	Move	X, A	(*Mem*[X]<-*Mem*[A])
2.	Add	X, B	(*Mem*[X]<-*Mem*[X]+*Mem*[B])
3.	Move	C, A	(*Mem*[C]<-*Mem*[A])
4.	Sub	C, B	(*Mem*[C]<-*Mem*[C]-*Mem*[B])
5.	Mul	C, X	(*Mem*[C]<-*Mem*[C]**Mem*[X])

Zakładając współczesne pamięci –> słowo procesora 32 bitowe. Dwa adresy w instrukcji –> Dwa słowa adresu.

Łącznie w programie 10 cykli dostępu do pamięci programu (2*5 instrukcji) i 13 cykli dostępu do pamięci danych (pobranie argumentów).

Zbiór instrukcji - przykład

Operacja: $c=a^2-b^2=(a+b)*(a-b)$

C. Wprowadzenie rejestru wewnętrznego - akumulatora:

1.	Load	A	(*ACC<-Mem*[A])
2.	Add	B	(*ACC<-ACC+Mem*[B])
3.	Store	X	(*Mem*[X]*<-ACC*)
4.	Load	A	(*ACC<-Mem*[A])
5.	Sub	B	(*ACC<-ACC-Mem*[B])
6.	Mul	X	(*ACC<-ACC*Mem*[X])
7.	Store	C	(*Mem[C]<-ACC*)

Jedna instrukcja –> jedno odwołanie do pamięci.

Łącznie w programie 7 cykli dostępu do pamięci programu (1*7 instrukcji) i 7 cykli dostępu do pamięci danych (pobranie argumentów i przechowanie wyników).

Zbiór instrukcji - przykład

Operacja: $c=a^2-b^2=(a+b)*(a-b)$

D. Wprowadzenie bloku rejestrów:

1.	Load	R1, A	(*RF[1]<-Mem*[A])
2.	Load	R2, B	(*RF[2]<-Mem*[B])
3.	Move	R3, R1	(*RF*[3]*<-RF[1]*)
4.	Add	R1, R2	(*RF[1]<-RF*[1]+RF[2])
5.	Sub	R3, R2	(*RF[3]<-RF[3]-RF[2]*)
6.	Mul	R1, R3	(*RF[1]<-RF[1]*RF*[3])
7.	Store	C, R1	(*Mem[C]<-RF[1]*)

Jedna instrukcja –> jedno odwołanie do pamięci.

Łącznie w programie 7 cykli dostępu do pamięci programu (1*7 instrukcji) i 3 cykle dostępu do pamięci danych (pobranie argumentów i przechowanie wyników).

Strategia współczesna

Połączenie trybów A i D: 6 pobrań instrukcji i 3 odwołania do pamięci w celu pobrania argumentów i zapisu wyników.

1.	Load	R1, A	(RF[1]<-Mem[A])
2.	Load	R2, B	(RF[2]<-Mem[B])
4.	Add	R3, R1, R2	(RF[3]<-RF[1]+RF[2])
5.	Sub	R4, R1, R2	(RF[4]<-RF[1]-RF[2])
6.	Mul	R5, R3, R4	(RF[5]<-RF[3]*RF[4])
7.	Store	C, R5	(Mem[C]<-RF[5])

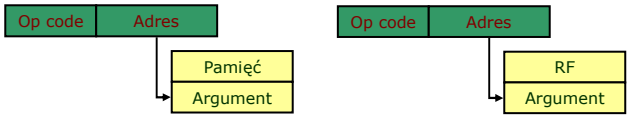
13

Tryby adresowania

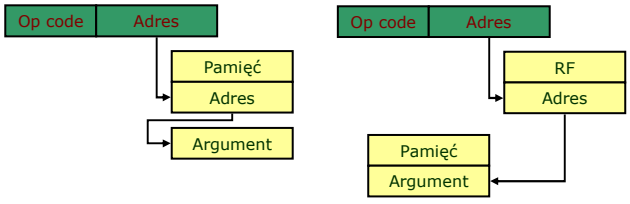
Wymuszone

Natychmiastowe

Bezpośrednie



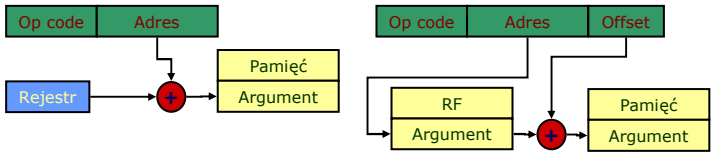
Pośrednie



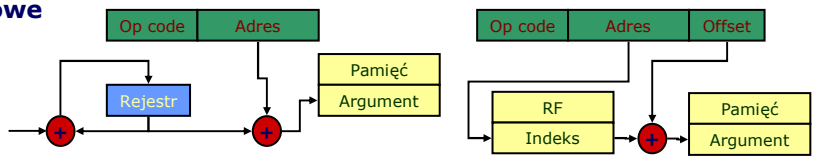
14

Tryby adresowania

Względne



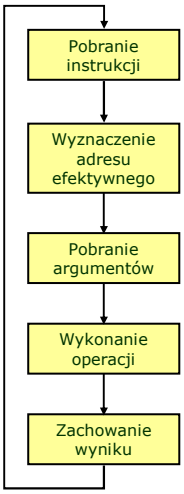
Indeksowe



15

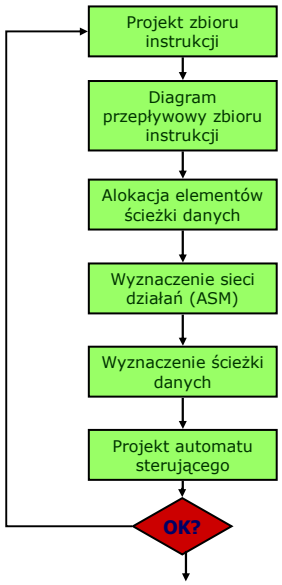
Cykl rozkazowy

Ogólny cykl rozkazowy zawiera 5 elementów:



16

Proces projektowania procesora



17

Projekt listy rozkazów

Kompromis pomiędzy efektywnością i wielkością programu oraz kosztem i wydajnością procesora.

Complex Instruction Set Computer (processor)
Duża liczba wydajnych instrukcji (wiele typów instrukcji, pól adresowych, trybów adresowania, operacji) -> mały program, rozbudowana ścieżka danych z wieloma elementami i połączeniami pomiędzy nimi

Reduced Instruction Set Computer (processor)
Prostsza implementacja, mniej elementów w ścieżce danych -> większy program, dłuższa sekwencja prostszych instrukcji.

Czas wykonania programu niekoniecznie jest krótszy w przypadku CISC.

RISC – możliwość implementacji potokowej ścieżki danych

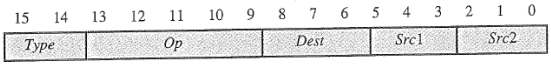
18

Projekt przykładowej listy rozkazów

Założenia:

- **procesor 16 bitowy,**
- **pamięć 64k słów,**
- **ograniczenie słów rozkazu do dwóch.**

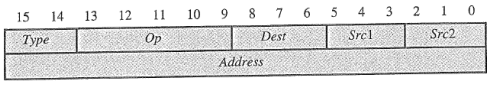
Instrukcje rejestrowe (arytmetyczne, logiczne, kopiowania i przesuwania bitowego):



Name	Action
Op Dest, Src1, Src2	*RF(Dest) ← RF(Src1) Op RF(Src2)*

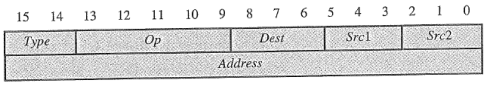
Projekt przykładowej listy rozkazów

Instrukcje pamięciowe (odczyt i zapis):



Name	Action
L imm Dest	*RF[Dest] ← Address*
L dir Dest	*RF[Dest] ← Mem[Address]*
L rel Dest, Src2	*RF[Dest] ← Mem[RF[Src2] + Address]*
L in Dest	*RF[Dest] ← Mem[Mem[Address]]*
S dir Src1	*Mem[Address] ← RF[Src1]*
S rel Src1, Src2	*Mem[RF[Src1] + Address] ← RF[Src1]*
S in Src1	*Mem[Mem[Address]] ← RF[Src1]*

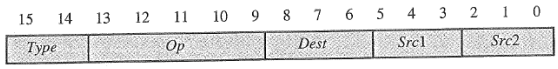
Instrukcje sterujące:



Name	Action
Jump Address	*PC ← Address*
Brel Address	*PC ← PC + 1 if Status[rel] = 0* *PC ← Address if Status[rel] = 1*
Call Address, Src1	*Mem[Src1] ← PC + 1; PC ← Address;* *RF[Src1] ← RF[Src1] + 1*
Return	*RF[Src1] ← RF[Src1] – 1; PC ← Mem[Src1]*

Projekt przykładowej listy rozkazów

Instrukcje pozostałe:



Name	Action
No-op	*Do nothing*
Clear Dest	*RF[Dest] ← 0*
Lstat Src1, Src2	*Status ← R[Src1] ≶ RF[Src2]*
Sstat Dest	*Status[Dest] ← 1*
Rstat Dest	*Status[Dest] ← 0*

Diagram przepływowy – opisuje cykl wykonania wszystkich instrukcji procesora, nie uwzględnia szczegółów architektury ani zależności czasowych.

Diagram przepływowy listy rozkazów procesora CISC (1):

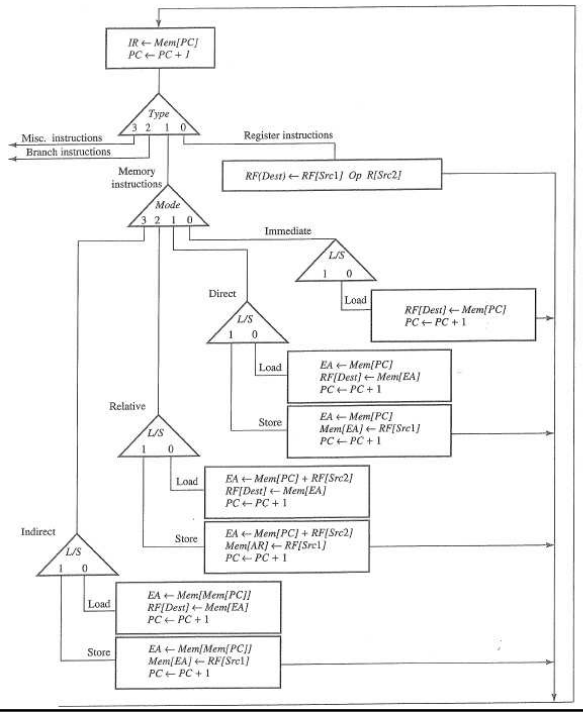
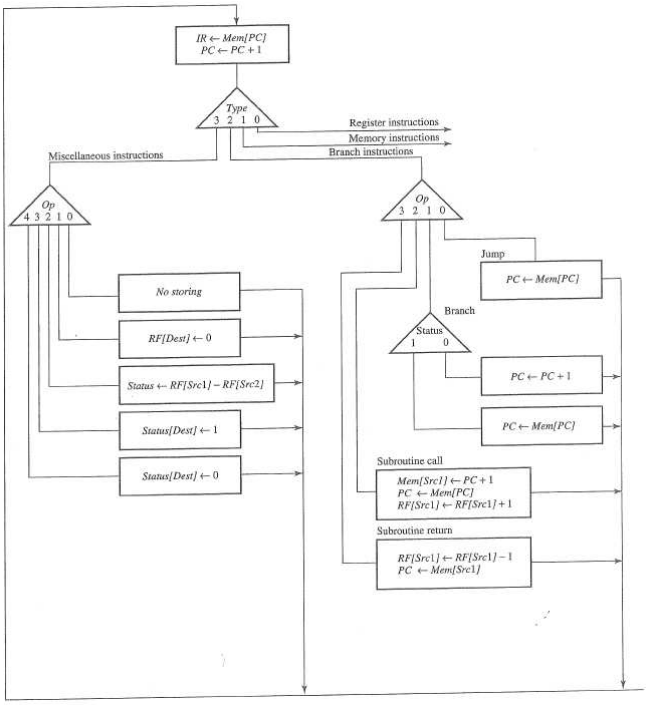
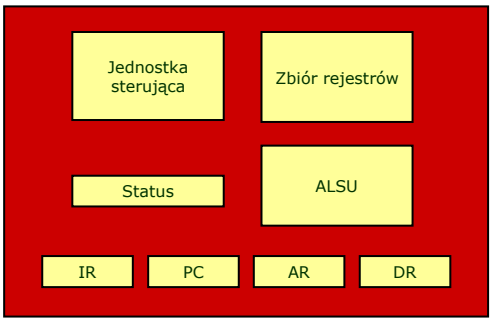


Diagram przepływowy listy rozkazów procesora CISC (2):



Przydzielanie zasobów procesora:



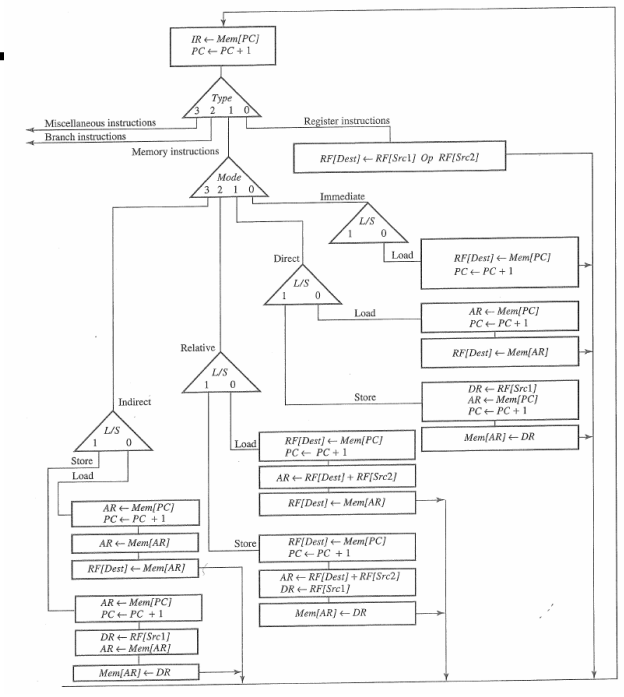
Z powodu niedopasowania pamięci wprowadzamy rejestry pomocnicze, umożliwiające obliczenie adresu efektywnego w trakcie 1 cyklu zegarowego:
AR – address register
DR – data register

Harmonogramowanie

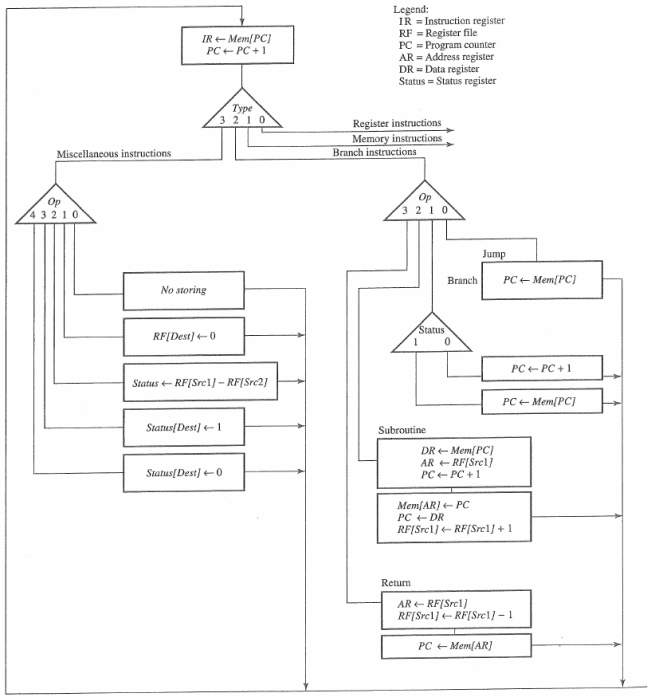
Detekcja konfliktów danych i zasobów:
- możliwość odczytu starej wartości z rejestru i zapis nowej do tego samego rejestru w trakcie cyklu zegarowego,
- brak możliwości odwrotnej: zapis danej i odczyt nowej wartości w tym samym takcie zegara.

W przypadku pamięci brak możliwości jednoczesnego (w tym samym takcie zegara) zapisu i odczytu danych.
To samo dotyczy dowolnego modułu funkcjonalnego poza pamięcią.

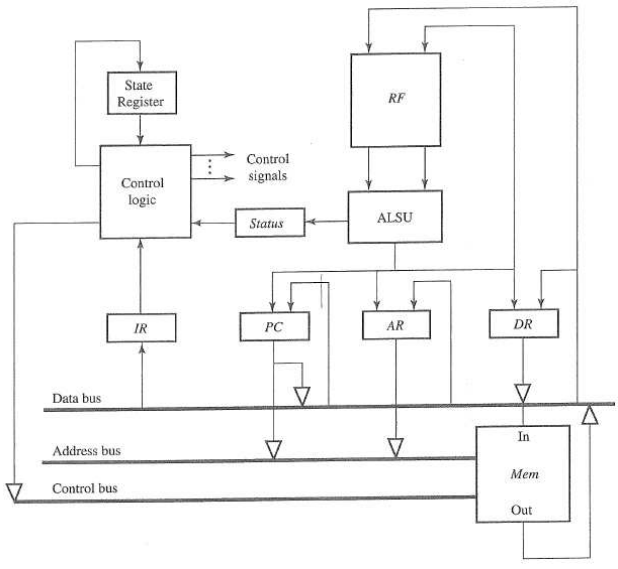
Sieć działań procesora (1):



Sieć działań procesora (2):



Schemat procesora na podstawie sieci działań:



28