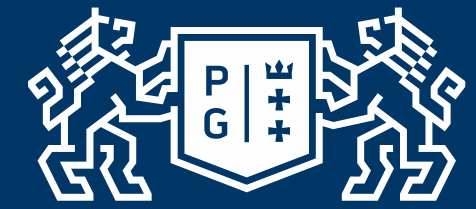


Architektura komputerów

sem. zimowy 2024/2025

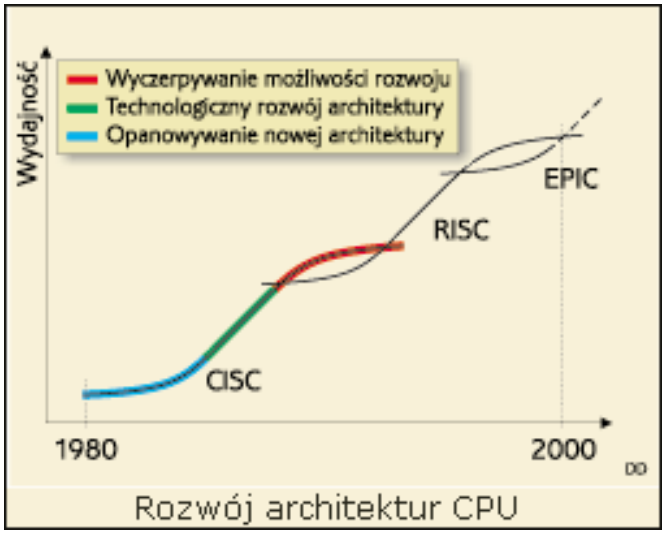
cz. 6

Tomasz Dziubich

POLITECHNIKA GDAŃSKA

Trochę historii

- Pierwsze procesory on-chip (IBM, Intel) ~1965, 1971
- Procesory RISC (HP, Patterson) ~ 1985
- Procesory EPIC (Intel) ~ 2000

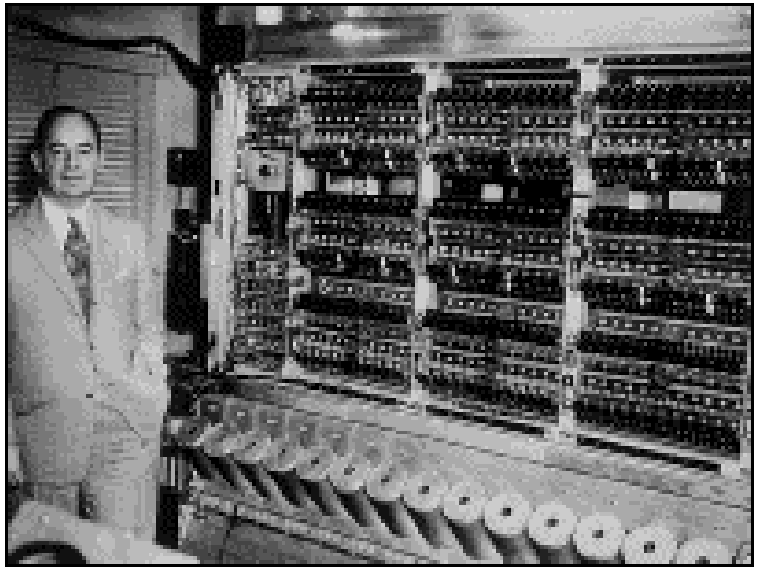


CISC

- Complex Set Instruction Computer – Komputery o złożonym zbiorze instrukcji
- Brak precyzyjnej definicji
  - Duża liczba rozkazów (100 – 250)
  - Rozkazy realizujące specjalizowane zadania (MOVS itd.)
  - Duża liczba trybów adresowania (5 – 20)
  - Zróżnicowana długość rozkazów
  - Zróżnicowany czas wykonania rozkazów
  - Mikroprogramowalna jednostka sterująca

VAX 11/780 (IBM)

- 304 Instrukcje
- 16 trybów adresowania
- 12 długości rozkazów



POLITECHNIKA GDAŃSKA

Wykorzystanie rozkazów

Procesory 16 bitowe	Motorola MC68020	Intel 80286
Typ rozkazu	**Średnie użycie**	**Średnie użycie**
Przesyłanie danych	**46,3%**	**43%**
Skoki, wywołania	**26,6%**	**15%**
Arytmetyczne	**14,1%**	**15%**
Porównania	**10,4%**	**10%**
Logiczne	**1,6%**	**7%**

POLITECHNIKA GDAŃSKA

Procentowe wykorzystanie rozkazów w architekturze x86

Instrukcje	Średnia
load	22
conditional branch	20
compare	16
store	12
add	8
and	6
sub	5
move register-register	4
call	1
return	1
Razem	**95**

POLITECHNIKA GDAŃSKA

Wykorzystanie odwołań do danych

- Większość argumentów wykorzystywanych w językach wysokiego poziomu to zmienne pojedyncze (skalarne)
- 80% wszystkich zmiennych stanowią zmienne lokalne
- Liczba parametrów podprogramów — średnio 5
- Liczba zmienny lokalnych podprogramów — średnio 7
- Poziom zagnieżdżenia wywołań podprogramów — średnio 6

RISC

- Reduced Set Instruction Computer – Komputery o zredukowanym zbiorze instrukcji
- Brak precyzyjnej definicji
  - Większość rozkazów wykonywana w jednym cyklu rozkazowym
  - Stosunkowo niewiele trybów adresowania
  - Łatwe do zdekodowania, stałej długości formaty rozkazów
  - Obszerny zbiór rejestrów uniwersalnych
  - Dostęp do pamięci ograniczony do rozkazów STORE/LOAD

RISC (2)

- Rozkazy działają zazwyczaj na argumentach w rejestrach, a nie w pamięci
- Układowo zrealizowana jednostka sterująca
- Przetwarzanie odbywa się w sposób potokowy
- Kompilatory dla RISC’ów mają duże możliwości optymalizacyjne

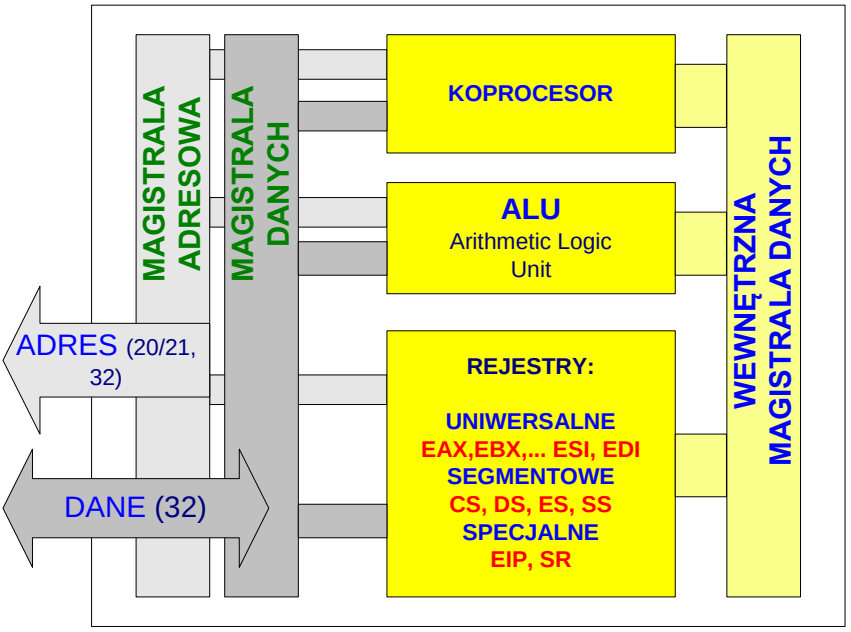
POLITECHNIKA GDAŃSKA

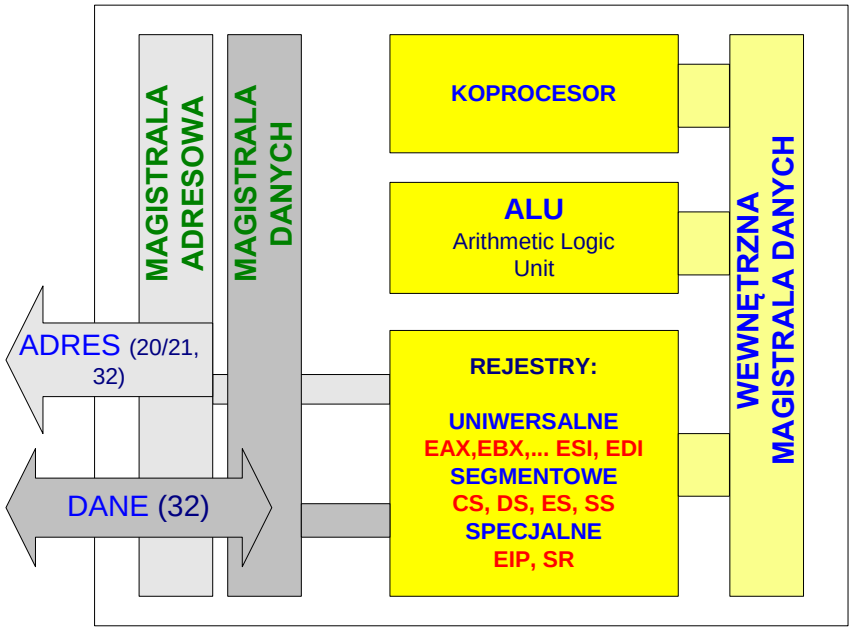
Porównanie RISC i CISC

- Rejestry wewnętrzne
- Formaty rozkazów
- Komunikacja z pamięcią
- Przekazywanie parametrów
- Jednostka sterująca
- Przetwarzanie potokowe
- Architektury superskalarne i superpotokowe
- Wydajność mikroprocesorów

POLITECHNIKA GDAŃSKA

Architektura RISC i CISC





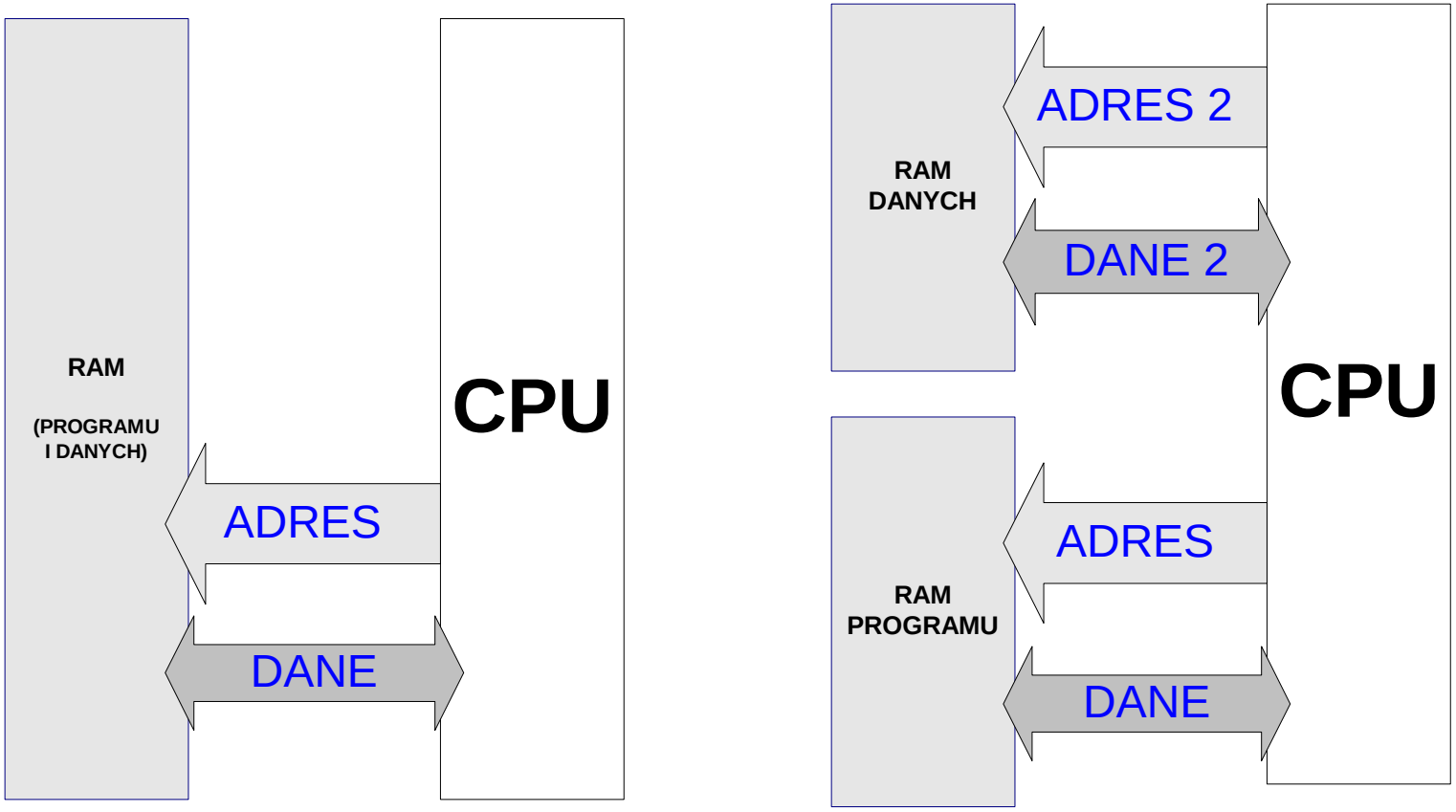
POLITECHNIKA GDAŃSKA

Architektura harwardzka

- Wąskie gardło klasycznej architektury von Neumanna to konieczność pobierania równocześnie instrukcji (kodu) oraz argumentów instrukcji (danych) z tej samej pamięci
- Architekturę (komputera) harwardzką wyróżnia rozdzielenie pamięci operacyjnej dla rozkazów i danych (dwie oddzielne magistrale)

POLITECHNIKA GDAŃSKA

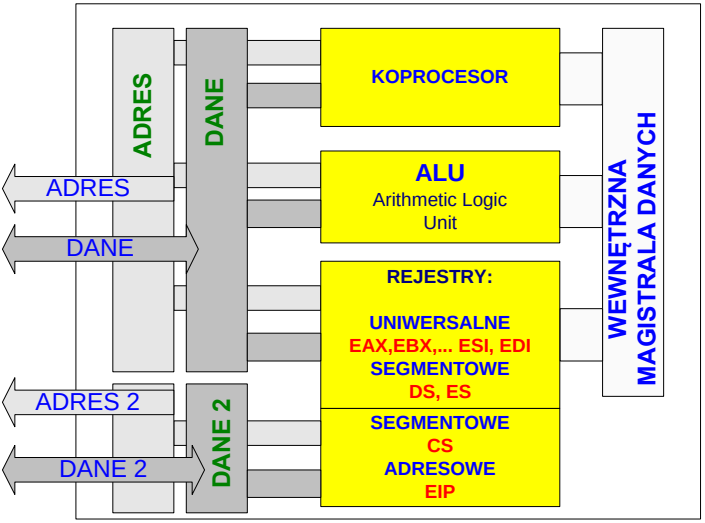
Architektura harwardzka

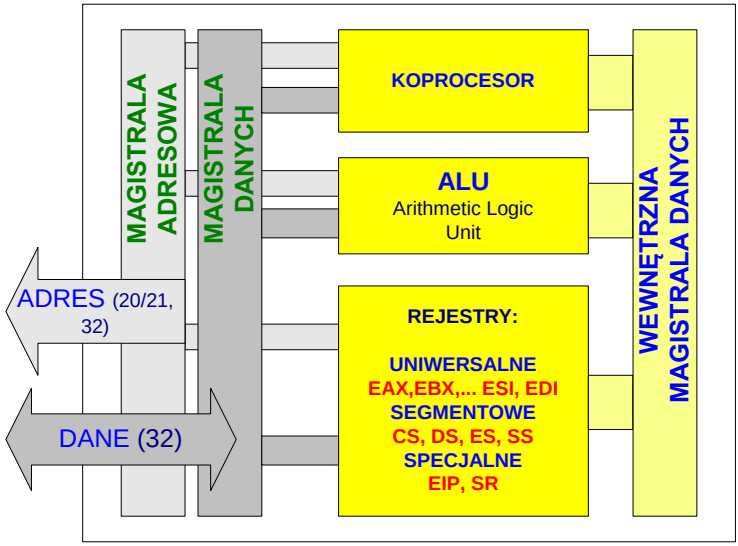


POLITECHNIKA GDAŃSKA

Architektura harwardzka

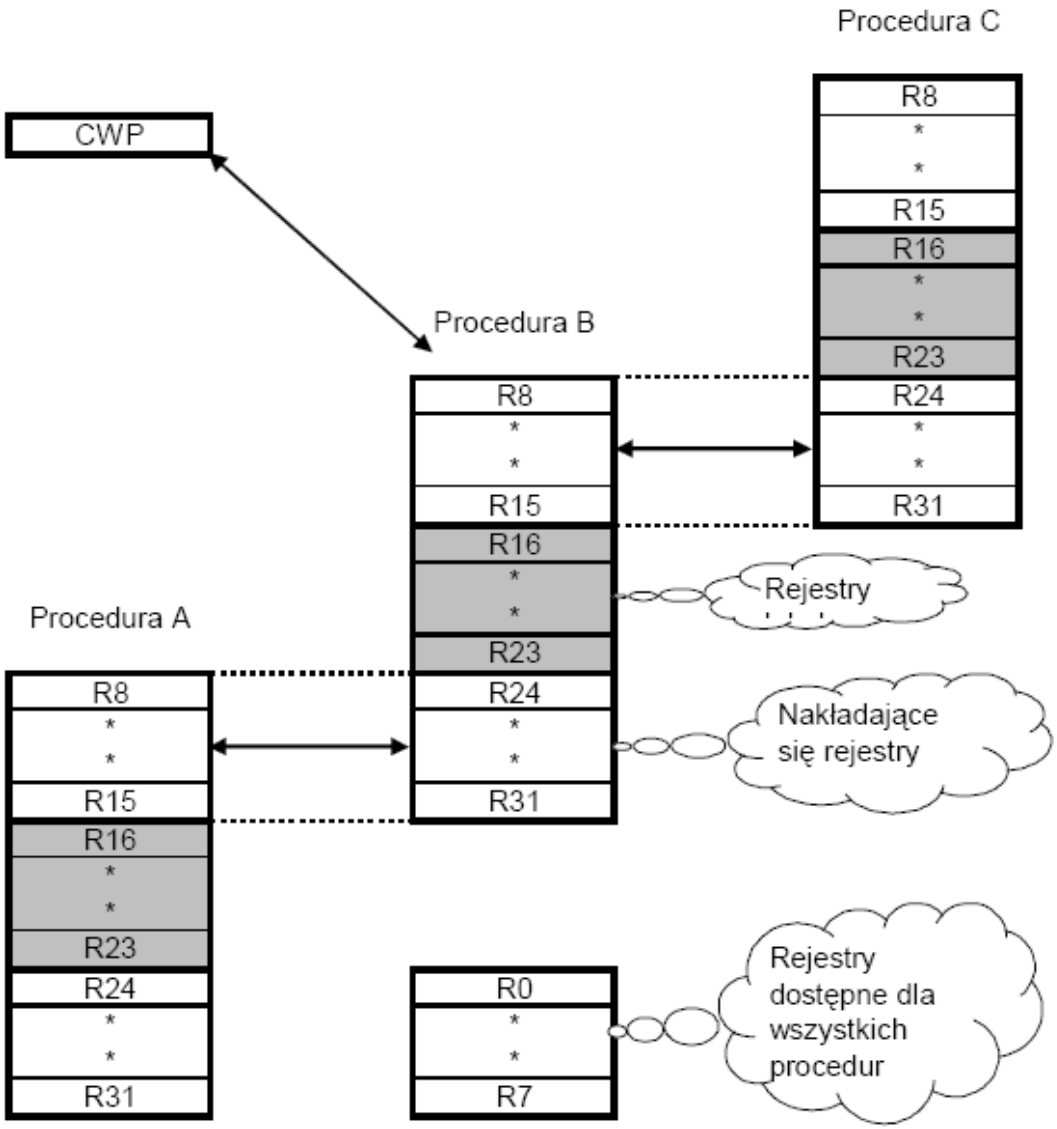
- Architektura ta ma zastosowanie dla obu rodzin procesorów, jednak do tej pory były wykorzystywana w RISC'ach





POLITECHNIKA GDAŃSKA

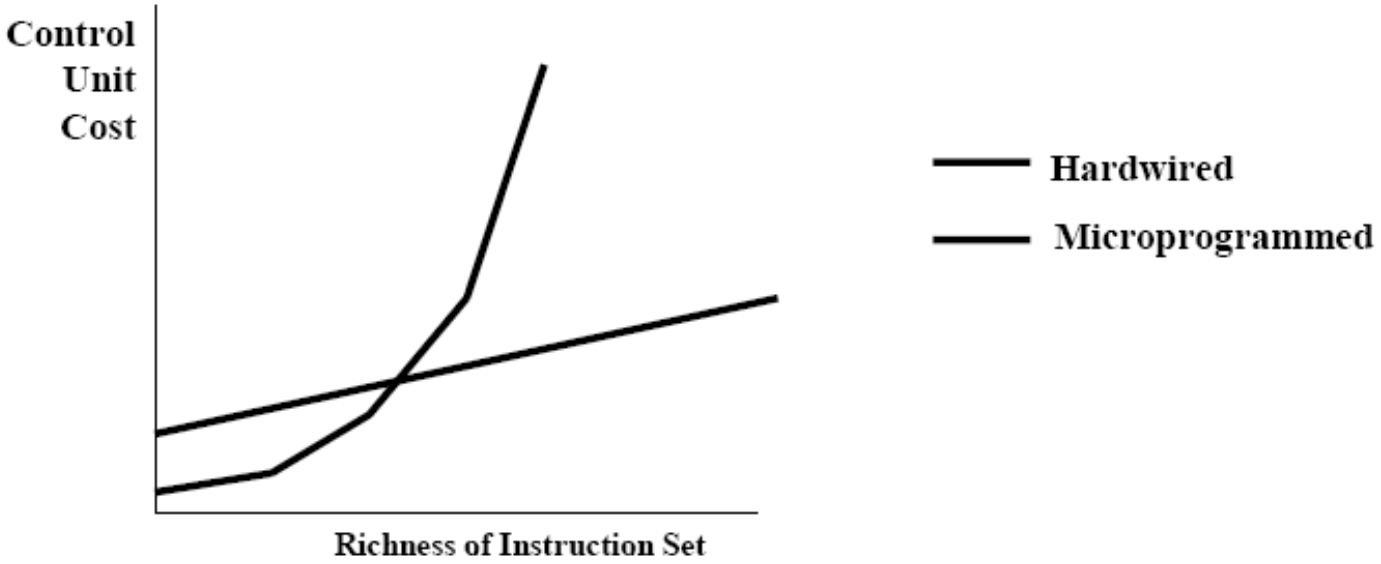
Przekazywanie paramterów (RISC)



POLITECHNIKA GDAŃSKA

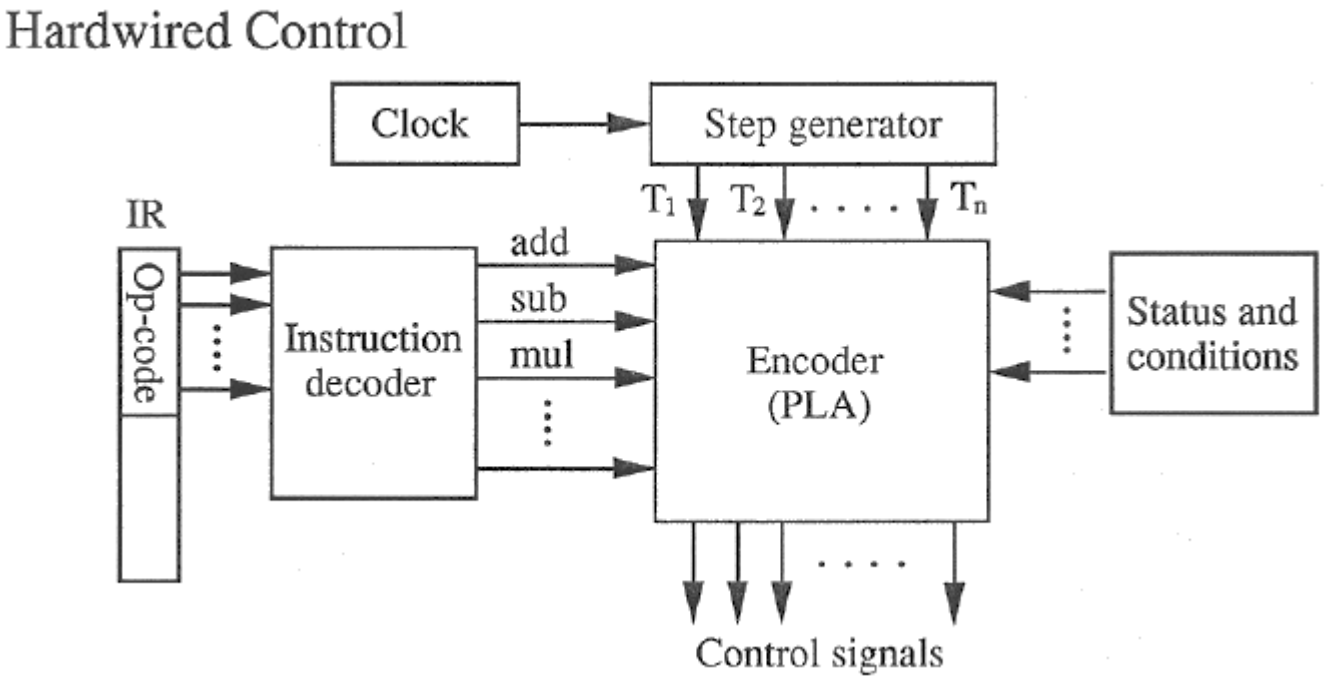
Jednostka sterująca

- Mikroprogramowa
- Układowa



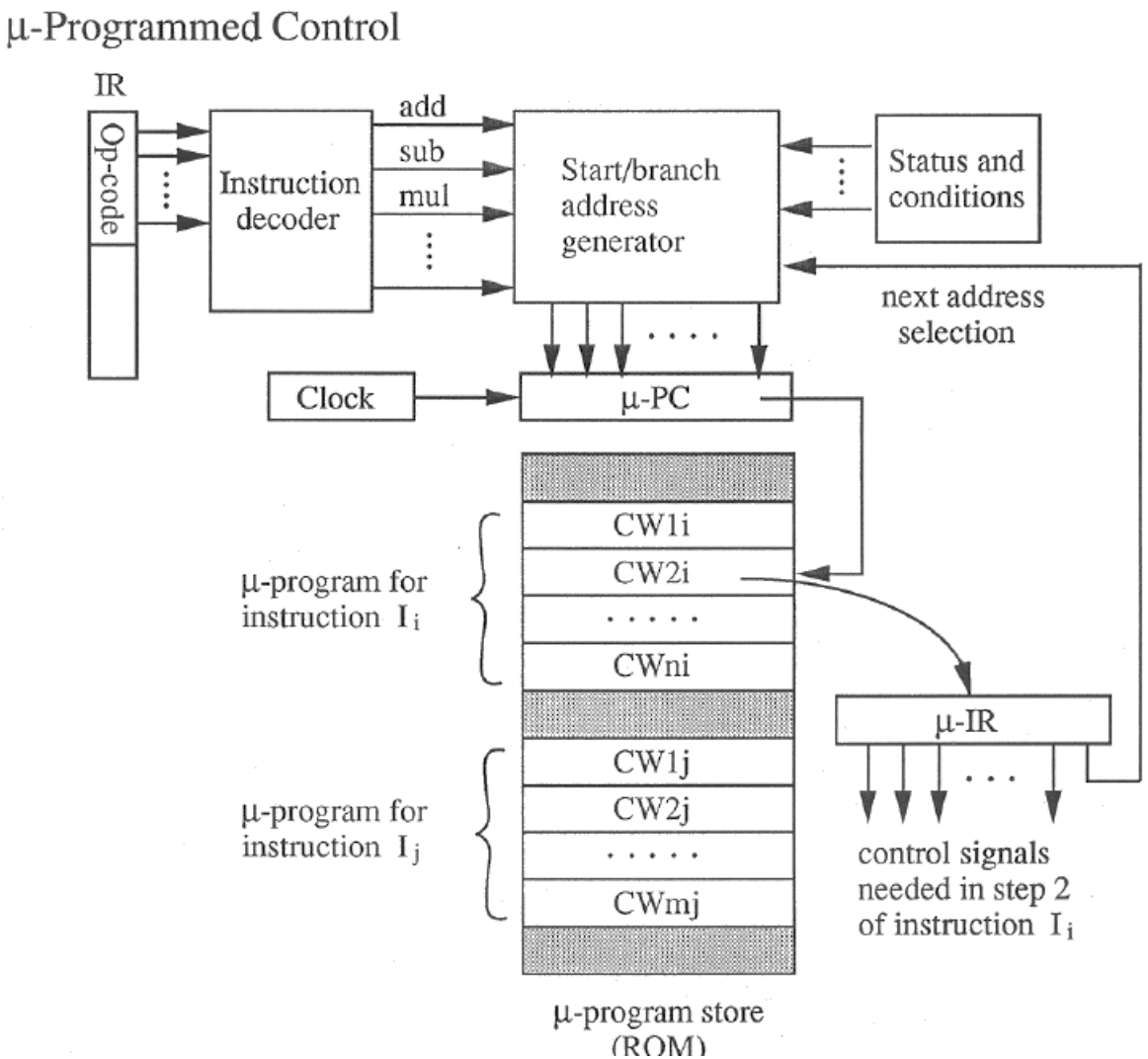
POLITECHNIKA GDAŃSKA

Budowa układowej jednostki sterującej



POLITECHNIKA GDAŃSKA

Budowa mikroprogramowej JS



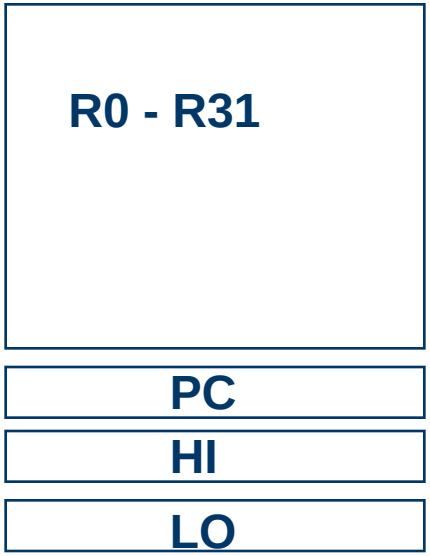
Formaty rozkazów

- Wąskie gardło klasycznej architektury von Neumanna to konieczność pobierania równocześnie instrukcji (kodu) oraz argumentów instrukcji (danych) z tej samej pamięci
- Architekturę (komputera) harwardzką wyróżnia rozdzielenie pamięci operacyjnej dla rozkazów i danych (dwie oddzielne magistrale)

POLITECHNIKA GDAŃSKA

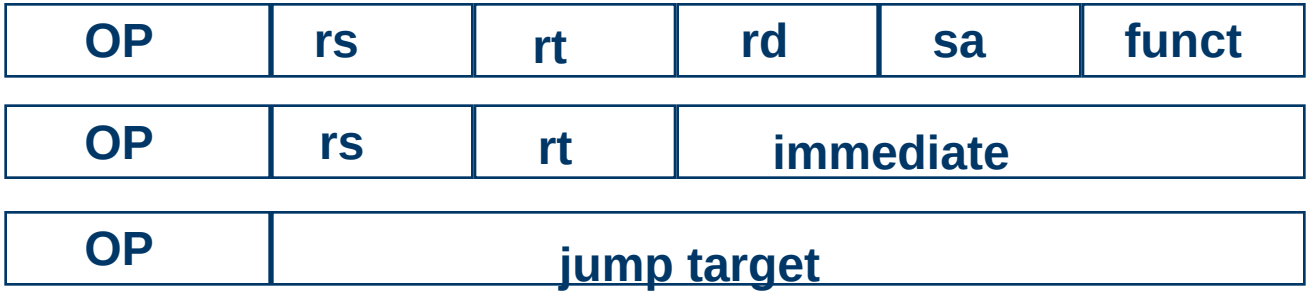
MIPS R3000 ISA (1988)

Rejestry



- Kategorie instrukcji
  - Load/Store
  - Computational
  - Jump and Branch
  - Floating Point
    - coprocessor
  - Memory Management
  - Special

Trzy formaty instrukcji, wszystkie o rozmiarze 32 bitów




```
add $1,$2,$3        ; $1 = $2 + $3 (signed)
```

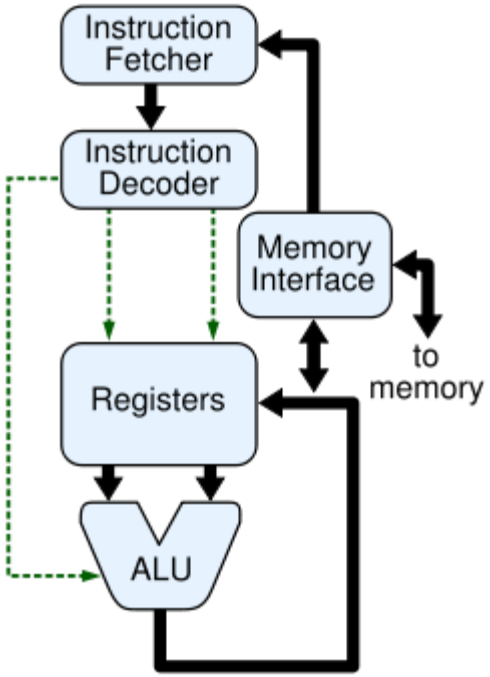
POLITECHNIKA GDAŃSKA

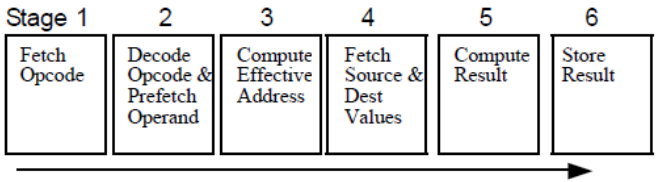
Przetwarzanie potokowe

W wykonaniu każdej instrukcji przez procesor można wyróżnić kilka faz. Typowo wyróżnia się następujące fazy:

- FETCH -pobranie instrukcji
- DECODE - dekodowanie obrazu binarnego instrukcji
- READ - odczyt argumentów instrukcji z rejestrów procesora lub z pamięci
- EXECUTE - wykonanie operacji arytmetycznej lub logicznej
- WRITE - zapis wyniku do rejestru lub pamięci

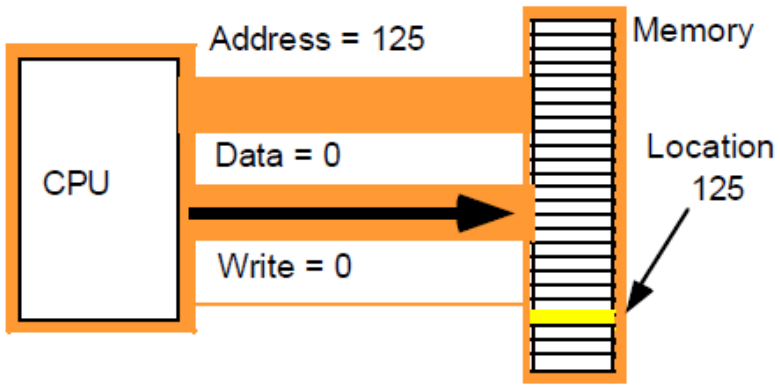
Należy zauważyć, że dla wielu instrukcji niektóre z wymienionych faz są zbędne, np. instrukcja przesłania międzyrejestrowego nie wymaga użycia jednostki arytmetyczno-logicznej

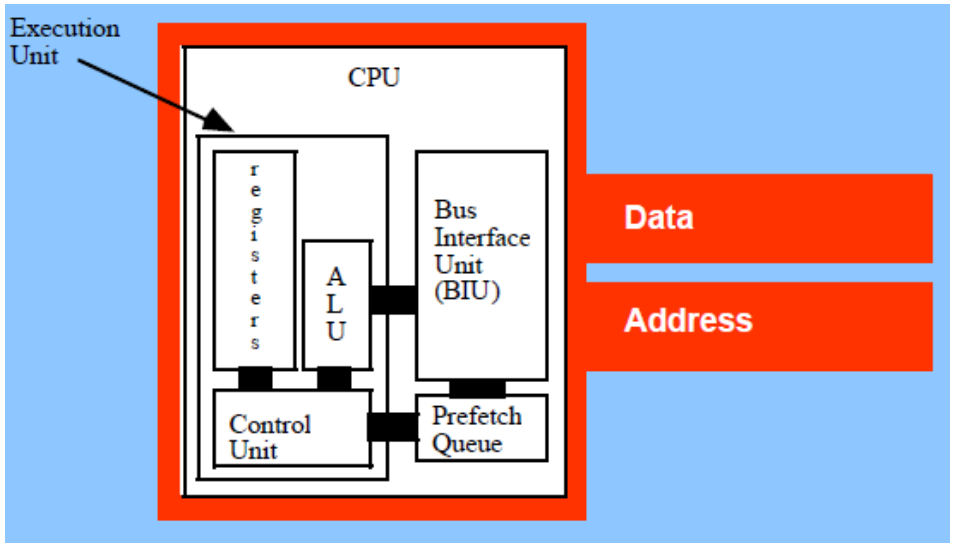


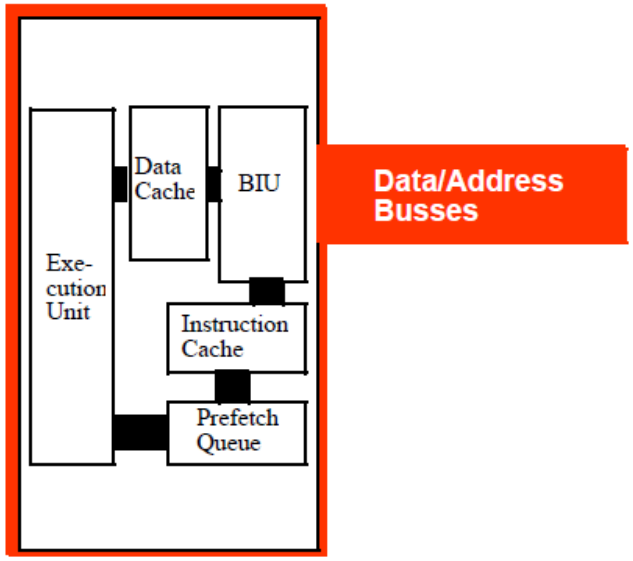


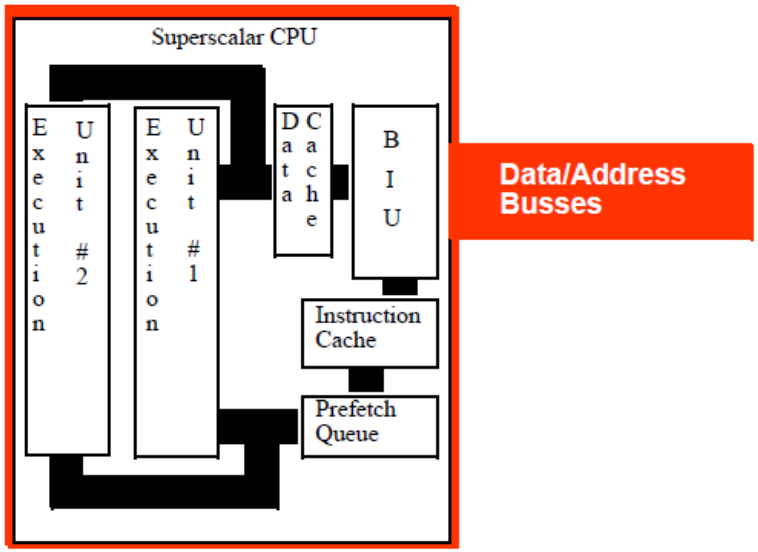
POLITECHNIKA GDAŃSKA

Bloki funkcjonalne procesora



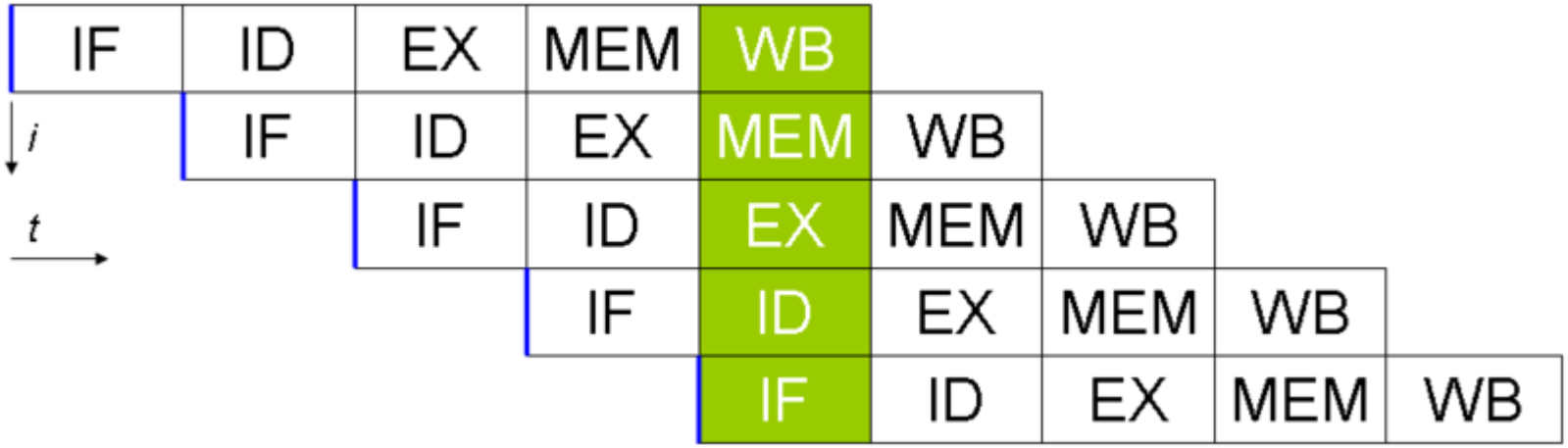






Przetwarzanie potokowe

Przetwarzanie potokowe



add EAX, 1234h = 05 34 12 00 00

krok	ADRES	DANE
1	IP	**Fetch:** pobranie kodu instrukcji
2	-	**Decode** (rozpoznanie instrukcji)
3	IP+1	**Read:** operand (00 00 12 34h)
4	-	**Execute** (wykonywanie instrukcji)
5	xxxx	**Write:** wynik dodawania
6	IP+5	Fetch: następna instrukcja
7	...	...

HIPERPOTOK PROCESORA Intel Pentium 4

Przetwarzanie potokowe

rok	ADD [1234h],EBX	ADD ECX,10	MOV BX,[22h]	MOV DX,[22h]	MOV DX,[22h]
1	FETCH				
2	DECODE	FETCH			
3	READ	DECODE	FETCH		
4	EXECUTE	READ	DECODE	FETCH	
5	WRITE	EXECUTE	READ	DECODE	FETCH
6	FETCH (nast. instr.)	WRITE	EXECUTE	READ	DECODE
7	DECODE	FETCH (nast. instr.)	WRITE	EXECUTE	READ
8		DECODE	FETCH (nast. instr.)	WRITE	EXECUTE
9			DECODE	FETCH (nast. instr.)	WRITE
10				DECODE	FETCH (nast. instr.)

Potok o „głębokości” 5:

Wykonywanie każdej instrukcji trwa 5 cykli, ale *efektywnie* tylko jeden.

POLITECHNIKA GDAŃSKA

Przetwarzanie potokowe

Zakłócenia w PP nazywa się hazardami

- realizacja niektórych etapów może powodować konflikty dostępu do pamięci;
- jeśli czasy trwania poszczególnych etapów mogą być niejednakowe, to na różnych etapach wystąpi pewne oczekiwanie;
- w programie występują skoki (rozgałęzienia) warunkowe, które mogą zmienić kolejność wykonywania instrukcji;
- wystąpienie przerwania sprzętowego lub wyjątku procesora stanowi zdarzenie nieprzewidywalne i również pogarsza przetwarzanie potokowe;
- niektóre rozkazy wymagają dodatkowych cykli (np. do ładowania danych);
- czasami rozkazy muszą oczekiwać z powodu zależności od nie zakończonych poprzednich rozkazów
- system musi zawierać rozwiązania zapobiegające tego rodzaju konfliktom;

Przetwarzanie potokowe

Podstawowe problemy: oczekiwanie na wyniki

- odczyt po zapisie
- zapis po odczycie
- zapis po zapisie

rok	ADD EAX,1234h	ADD EAX,EBX
1	FETCH	
2	DECODE	FETCH
3	READ	DECODE
4	EXECUTE	READ
5	WRITE	EXECUTE
6	FETCH (nast. instr.)	WRITE
7	DECODE	FETCH (nast. instr.)
8		DECODE
9		

rok	ADD EAX,1234h	ADD EAX,EBX
1	FETCH	
2	DECODE	
3	READ	
4	EXECUTE	FETCH
5	WRITE	DECODE
6	FETCH (nast. instr.)	READ
7	DECODE	EXECUTE
8		WRITE
9		FETCH (nast. instr.)
10		DECODE

Rozwiązanie

- Szeregowanie statyczne (odpowiednia kompilacja usuwa 70% zależności)
- Szeregowanie dynamiczne (sprzętowe) – prekursor Out of Order
- Technika notowania (scoreboarding)

Przetwarzanie potokowe

Podstawowe problemy: rozgałęzienia programu

rok	ADD [1234h],EBX	JC +10	MOV BX,[22h]	MOV DX,[22h]	MOV DX,[22h]
1	FETCH				
2	DECODE	FETCH			
3	READ	DECODE	FETCH		
4	EXECUTE	READ	DECODE	FETCH	
5	WRITE	EXECUTE	READ	DECODE	FETCH
6	FETCH (nast. instr.)	WRITE	EXECUTE	READ	DECODE
7	DECODE	FETCH (nast. instr.)	WRITE	EXECUTE	READ
8		DECODE	FETCH (nast. instr.)	WRITE	EXECUTE
9			DECODE	FETCH (nast. instr.)	WRITE
10				DECODE	FETCH (nast. instr.)

rok	ADD [1234h],EBX	JC +10	MOV AX, 0	MOV DX,[22h]	MOV DX,[22h]
1	FETCH				
2	DECODE	FETCH			
3	READ	DECODE			
4	EXECUTE	READ			
5	WRITE	EXECUTE			
6	FETCH (nast. instr.)	WRITE			
7	DECODE	FETCH (nast. instr.)	FETCH		
8		DECODE	DECODE	FETCH	
9			READ	DECODE	FETCH
10			EXECUTE	READ	DECODE
11			WRITE	EXECUTE	READ
12			FETCH (nast. instr.)	WRITE	EXECUTE

Rozwiązanie

- Zwielokrotnianie strumieni
- Bufor pętli
- Przewidywanie statyczne (75%)
- Przewidywanie dynamiczne
  - Bit(y) historii
  - Tablica historii skoków (BTB)
- Kodowanie bez użycia skoków

Inne rozwiązanie

- Metoda predykatowa

```
if (x > 5) printf("x > 5\n");
else printf("x <= 5\n");

b1, b2 = cmp.gt x,5       // obliczenie zmiennych
                          // boolowskich b1 i b2 na
                          // podstawie porównania x i 5
(b1) callprintf "x > 5\n"   // instrukcja jest
                          // wykon. gdy b1=true
(b2) callprintf "x <= 5\n"          // instrukcja jest
                          // wykon. gdy b2=true
```

POLITECHNIKA GDAŃSKA

Inne rozwiązania

- Kodowanie bez użycia skoków

```
        mov eax, esi
        cmp esi, edi
        jae dalej
        mov eax, edi
dalej:  call wyswietl32
```

```
b = ESI > EDI
wynik = ESI & b + EDI & (~ b)
print (wynik)
```

```
mov edx, 0          ; zmienna b
cmp esi, edi
setg dl             ; wpisanie 1 do DL
                    ; jeśli ESI > EDI
neg edx             ; zmiana znaku liczby
                    ; kodowanej w U2
; jeśli liczba w ESI była większa od liczby w EDI, to w EDX
; (zmienna b) będą same jedynki, w przeciwnym razie
  same zera
and esi, edx
not edx
and edi, edx
or esi, edi
mov eax, esi
call wyswietl32
```

Inne rozwiązania

- Optymalizacja kodu

```
mov ax, K
add ax, M
add ax, N
add ax, P
mov J, ax
```

```
mov bx, K
mov ax, M
add bx, N
add ax, P
add ax, bx
mov J, ax
```

Rozwiązanie

1. r16 = call rand ( )
2. add r16 = r16, 2
3. mul r16 = r16, 4
4. st8 [r16] = 2
5. ld8 r17 = addr("x")

// wyznaczanie adresu "x"

6. ld8 r18 = [r17]

// ładowanie zmiennej x z pamięci

8. add r18 = r18, 1
9. st8 [r17] = r18

// zapisywanie x do pamięci

1. r16 = call rand ( )
2. ld8 r17 = addr("x")

// wyznaczanie adresu "x"

3. ld8 r18 = [r17]

// ładowanie zmiennej x z pamięci

4. add r16 = r16, 2
5. mul r16 = r16, 4
6. st8 [r16] = 2
7. add r18 = r18, 1
8. st8 [r17] = r18

// zapisywanie x do pamięci

POLITECHNIKA
GDAŃSKA

HISTORIA MĄDROŚCIĄ
PRZYSZŁOŚĆ WYZWANIEM