Wykład 14

Procesory 64-bitowe
Intel Itanium

dr hab. inż. Bartłomiej Zieliński, prof. PŚ.

Intel Itanium

Program:

- Główne założenia
- Podstawy architektury EPIC
- Rejestry, jednostki wykonawcze
- Format rozkazu
- Predykacja rozgałęzień

Intel Itanium

- Główne założenia
  - Nowe podejście
    - Wsparcie równoległości na poziomie rozkazów
      - Różne od architektur superskalarnych
    - Wykonanie predykatywne
      - Każdy rozkaz odnosi się do 1-bitowego predykatu
      - Rozkaz wykonany, gdy predykat=1

      → spekulatywne wykonanie skoków, instrukcje warunkowe

      → wynik ujawniany, gdy warunek jest określony

Intel Itanium

- Główne założenia
  - Nowe podejście
    - Spekulacja sterowaniem
      - Operacje ładowania przeniesione wcześniej
      - Wyjątek → obsługa tylko, gdy ładowanie było konieczne
    - Spekulacja danymi
      - Odczyt przed zapisem do odczytywanej komórki
      - Następnie sprawdzenie, czy dane są ważne
    - Programowe potokowanie
      - Równoległe wykonanie różnych iteracji pętli

Intel Itanium

- Architektura EPIC
  - *Explicit Parallel Instruction Computer*
    - („*komputer jawnie równoległy*”)
    - Równoległość na poziomie kodu widziana w programie
      - Nie trzeba jej określać podczas wykonywania rozkazów
    - Bardzo długie słowo instrukcji (→ architektura VLIW)
    - Predykacja skoków
    - Ładowanie spekulatywne

Intel Itanium

- Architektura EPIC
  - Explicit Parallel Instruction Computer
    - Sensowne wykorzystanie wielu mln tranzystorów
      - Nie jako pamięć podręczna
      - Bez zwiększania stopnia superskalarności
    - Rozkazy planowane statycznie przez kompilator
      - A nie dynamicznie przez procesor
      - Możliwości wykonania równoległego ustalane przez kompilator i użyte przez procesor podczas wykonania
      - Uproszczona struktura procesora
      - Lepsza optymalizacja
        - » Kompilator ma więcej zasobów potrzebnych do optymalizacji kodu

Intel Itanium

- Architektura EPIC
  - Duża liczba rejestrów
    - 128×64-b stałoprzecinkowe
    - 128×80-b zmiennoprzecinkowe
    - 64×1 predykaty

    → wysoki stopień równoległości
  - Duża liczba jednostek przetwarzających
    - Więcej niż 8
      - (procesory superskalarne → ok. 4)

Intel Itanium

- Architektura EPIC
  - Rejestry
    - Architektury superskalarne
      - Kilka widocznych rejestrów, wiele aliasów (przemianowywanie)
    - EPIC
      - Wiele widocznych rejestrów (jawna równoległość)
  - Jednostki wykonawcze
    - Zależy od liczby dostępnych tranzystorów
      - Np., jeśli 8 rozkazów można wykonać równolegle, ale są tylko 4 jednostki, to wykonanie zajmuje 2 cykle
      - Jeśli mamy 8 rozkazów i 8 jednostek → 1 cykl

Intel Itanium

- IA-64
  - 4 rodzaje jednostek wykonawczych
    - I(nteger) – stałoprzecinkowe
      - Arytmetyczno-logiczne, przesunięcia, porównania, rozkazy multimedialne stałoprzecinkowe
    - M(emory) – pamięciowe
      - Przesyły do i z pamięci, rozk. stałoprzecinkowe
    - B(ranch) – rozgałęzienia
      - Skoki, rozgałęzienia
    - F(loating) – zmiennoprzecinkowe
      - Operacje zmiennoprzecinkowe

Intel Itanium

- IA-64
  - 6 typów rozkazów

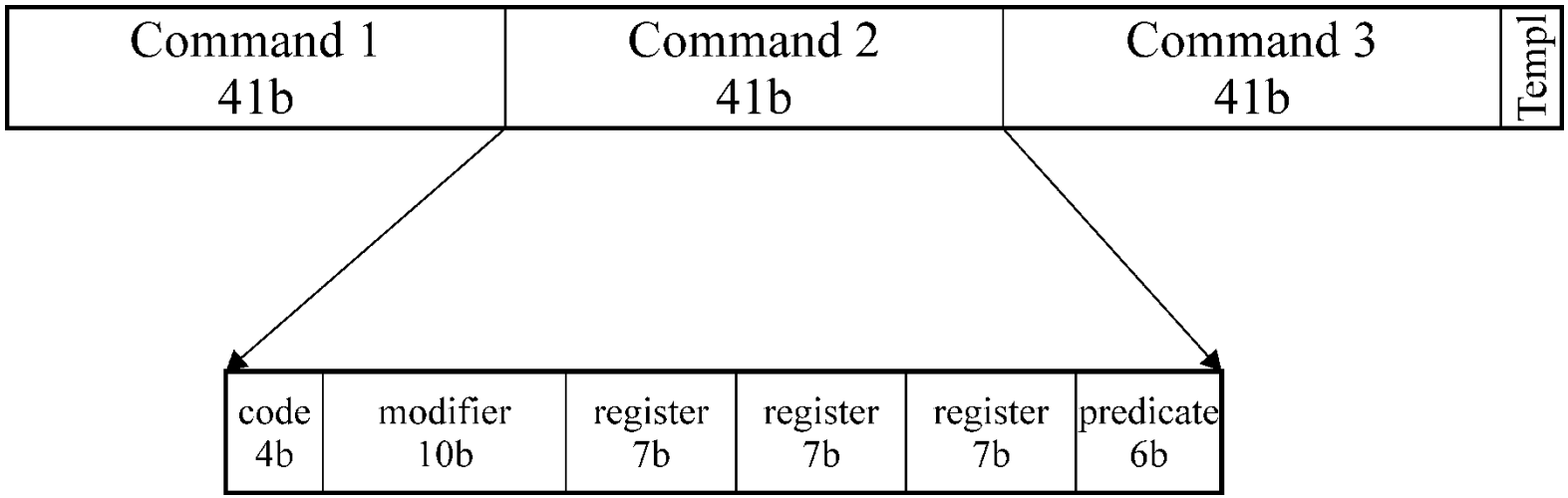
Symbol	Typ	Komentarz	Jedn.wyk.
A	Artym.-log.	Stałoprzecinkowe w ALU	I/M
I	Stałoprzecinkowe	Stałoprzecinkowe poza ALU	I
M	Pamięciowe	Operacje pamięciowe	M
F	Zmiennoprzecinkowe	Zmiennoprzecinkowe	F
B	Rozgałęzienia	Skoki	B
L+X	Rozszerzone	Rozszerzone	I/B

Intel Itanium

- Itanium – format rozkazu
  - 128-b wiązka (*bundle*) = 3 rozkazy + szablon
  - Procesor pobiera 1 lub więcej wiązek
  - Szablon (*template*)
    - Które rozkazy można wykonać równolegle
    - Grupa nie ograniczona do pojedynczej wiązki
  - Procesor przegląda wiele wiązek
    - Np. 8 rozkazów do wykonania równoległego
      - Kompilator umieszcza je w sąsiednich wiązkach
      - Odpowiednio ustawione pole szablonu

Intel Itanium

- Itanium – format rozkazu
  - Rozkazy w wiązce w dowolnej kolejności
    - Być może różnej niż w kodzie źródłowym
  - Szablon dzieli wiązkę na rozkazy zależne i niezależne
  - Nie trzeba wypełniać wiązki NOP-ami (→ VLIW!)



Intel Itanium

- Itanium – format rozkazu
  - Interpretacja głównego kodu rozkazu
    - W kontekście szablonu
    - W kontekście położenia w wiązce
  - Długość 41b
    - > 32b w RISC
    - < 118b µop Pentium Pro
    - Wynika z:
      - Dużej liczby rejestrów
      - Odniesienia do rejestru predykatu

Intel Itanium

- Itanium – format rozkazu
  - Kilka przykładowych szablonów

Szablon	Rozkaz 1	Rozkaz 2	Rozkaz 3
00, 01	M	I	I / I;;
02, 03	M	I;;	I / I;;
04, 05	M	L	X / X;;
08, 09	M	M	I / I;;

  - Rozkaz asemblerowy

```
[qp] mnem [.compl]  dst=srcs [;;]
```

Intel Itanium

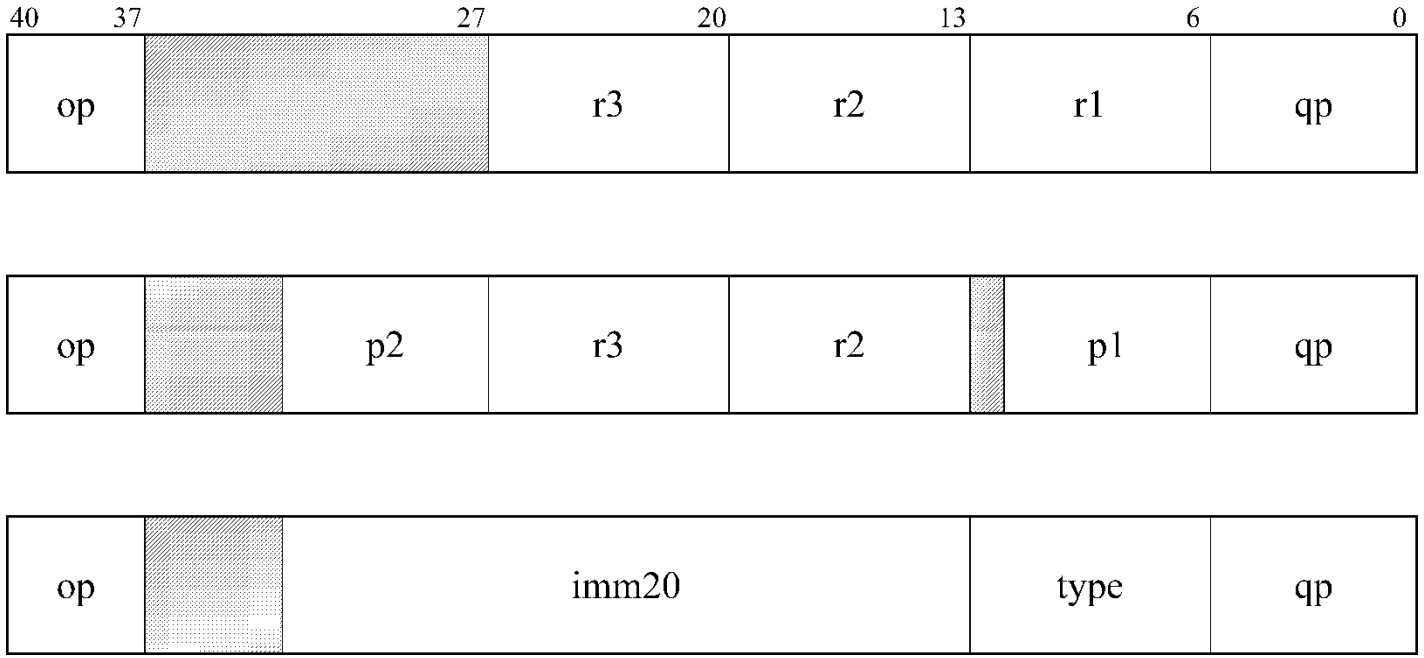
- Itanium – format rozkazu
  - Rozkaz asemblerowy

```
[qp] mnem [.compl]  dst=srcs [;;]
```

    - Qp – predykat kwalifikujący
      - QP0 – rozkaz bezwarunkowy
    - Mnem – nazwa rozkazu
    - Compl. – „uzupełniacz”
    - Dst – jeden lub więcej arg. docelowych
    - Srcs – arg. źródłowe (typowo 2)
    - ;; – granice grup

Intel Itanium

- Itanium – format rozkazu
  - Kilka przykładowych formatów
    - Arytmetyczno-logiczny
    - Porównanie
    - Skoki



Intel Itanium

- Itanium – format instrukcji
  - Kilka przykładowych rozkazów

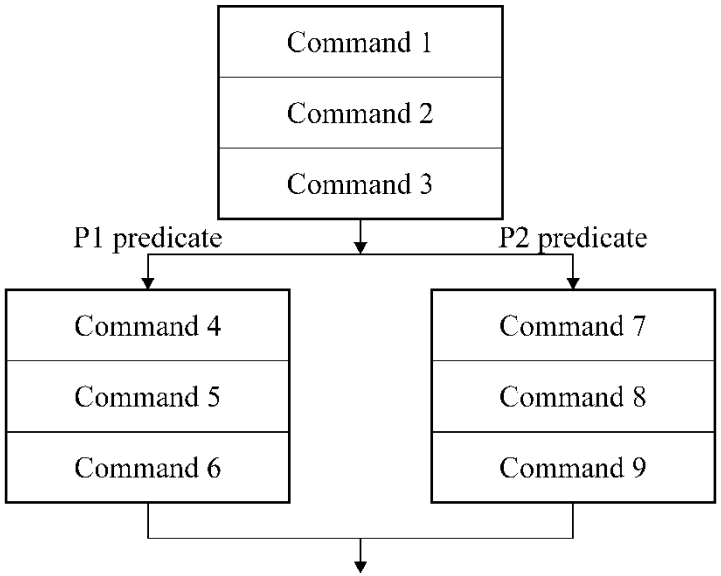
Rozkaz	Operacja	Komentarz
Add r5=r6,r7	R5←r6+r7	Dodawanie
Ld4 r2=[r3]	$R2_{31..0}$←M[r3]	Ładowanie 4B rej → pam.
(p4) add r1=r2,r3	P4=>r1←r2+r3	Dodaj gdy p4=1; else nop
(p6) add r8=-1,r32	P6=>r8←r32-1	Dodaj stałą gry p6=1; else nop
Cmp.eq p6,p5=r32,r0	P6←(r32=r0); p5←not p6	Sprawdź czy równe; ustaw p5,p6
Extr.u r31=r2,3,6	$R31_{5..0}$←$r2_{8..3}$	Ekstrakcja wybranych bitów
Cmp.eq.and p1,p2=r32,r33	(r32≠r33)=>p1=p2=0	Sprawdź czy równe

Intel Itanium

- Itanium – predyk***a***cja
  - Kompilator określa instrukcje do wykonania równolegle
  - Rozgałęzienia zastąpione wykonaniem warunkowym
    - Przewidywanie skoków zbędne

Intel Itanium

- Itanium – predyk**a**cja
  - 2 ścieżki → 2 predykaty
  - Równoległe wyk. ścieżek
    - Brak zależności między ścieżkami
  - Znany wynik porównania → usunięcie zbędnego wyniku
  - Kompilator może zmienić kolejność rozkazów



Command 1	Command 2	Command 3
Command 4	Command 7	Command 5
Command 8	Command 6	Command 9

Intel Itanium

- Itanium – predykacja
  - Przykład 1

```
if (r1==r2)
  r3 = r4+r5
else
  r6 = r4-r5;
```

```
        Cmp r1, r2
        Jne NotEq
        Mov r3, r4
        Add r3, r5
        Jmp Further
NotEq:
        Mov r6, r4
        Sub r6, r5
Further:
```

```
    cmp.eq p1, p2=r1, r2
(p1) add r3=r4, r5
(p2) sub r6=r4, r5
```

Przewidywanie skoków
Ryzyko błędnej prognozy

Równoległe wykonanie rozkazów
Przyjęcie właściwego wyniku

Intel Itanium

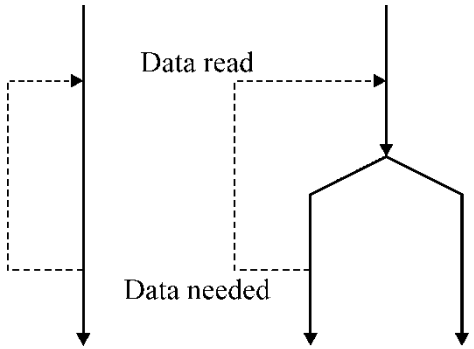
- Itanium – predykacja
  - Przykład 2

```
If (a && b)
  j++;
Else
  if (c)
    k++;
  else
    k--;
i++;
```

```
      cmp.eq p1,p2=0,a     // a=0?
(p2) cmp.eq pq, p3=0,b     // b=0?
(p3) add j=1,j             // j++
(p1) cmp.ne p4,p5=0,c      // c≠0?
(p4) add k=1,k             // k++
(p5) add k=-1,k            // k--
      add i=1,i            // i++
```

Intel Itanium

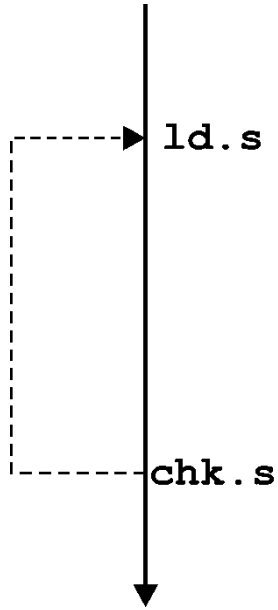
- Itanium – ładowanie spekulatywne
  - Odczyt danych z pamięci zanim będą potrzebne
    - Nie trzeba czekać na zakończenie ładowania
  - Ładowanie wcześniej w kodzie



  - *A jeśli ładowanie przed rozgałęzieniem?*
    - *Np. ładowanie predykatywne: odczyt, ale wypełnienie rejestrów dopiero gdy znamy wartość predykatu*
    - *Możliwy wyjątek (błąd adresu, stronicowania itp.)*

Intel Itanium

- Itanium – ładowanie spekulatywne
  - W IA-64 ładowanie oddzielone od wyjątku
    - Ld.s (*load speculative*): odczyt pamięci, wyjątek wykryty, ale nie zgłoszony
    - Chk.s (*check speculative*): w oryginalnym miejscu odczytu; może zależeć od predykatu; zgłasza wyjątek w razie potrzeby



    - Wykryto wyjątek ld.s → rejestr przyjmuje wartość NaT („Not a Thing” – „nic”)
    - Chk.s sprawdza NaT; jeśli =1, zgłoszenie wyjątku

Intel Itanium

- Itanium – ładowanie spekulatywne
  - przykład

```
Int funct (int *x) {
  if (x==NULL)
    return -1;
  else
    return (*x+7);
} /* funct */
```

```
Funct:
  ld8.s r1=[r32]
  cmp.eq p6,p5=r32,r0;;
(p6) add r8=-1,r0
(p6) br.ret
(p5) chk.s r1,err;;
  add r8=7,r1
  br.ret
……
Err:
  ld8 r1=[r32]
  add r8=7,r1
  br.rets
```

Intel Itanium

- Itanium – ładowanie wyprzedzające
  - Ładowanie przed zapisaniem ładowanej wartości
  - Sprawdzenie, czy dane wciąż aktualne

  - Ładowanie wpisuje adres do ALAT
    - *Advanced Load Address Table*
  - Zapis sprawdza ALAT
    - Usunięcie adresu z ALAT
  - Sprawdzenie sprawdza ALAT
    - Adres znaleziono → dane aktualne
    - Adresu nie znaleziono → dane nieaktualne, odczyt

Intel Itanium

- Itanium – ładowanie spekulatywne
  - Przykład

```
Add r3=4,r0;;
St4 [r32]=r3
Ld4 r2=[r33];;
Add r5=r2,r3
```

```
Ld4.a r2=[r33]
Add r3=4,r0;;
St4 [r32]=r3
Ld4.c r2=[r33];;
Add r5=r2,r3
```

```
Ld8.a r6=[r8]// advanced load; ALAT write
St8 [r4]=r12 // write and ALAT check
Ld8.c r6=[r8]// load check; ALAT check
```

Intel Itanium

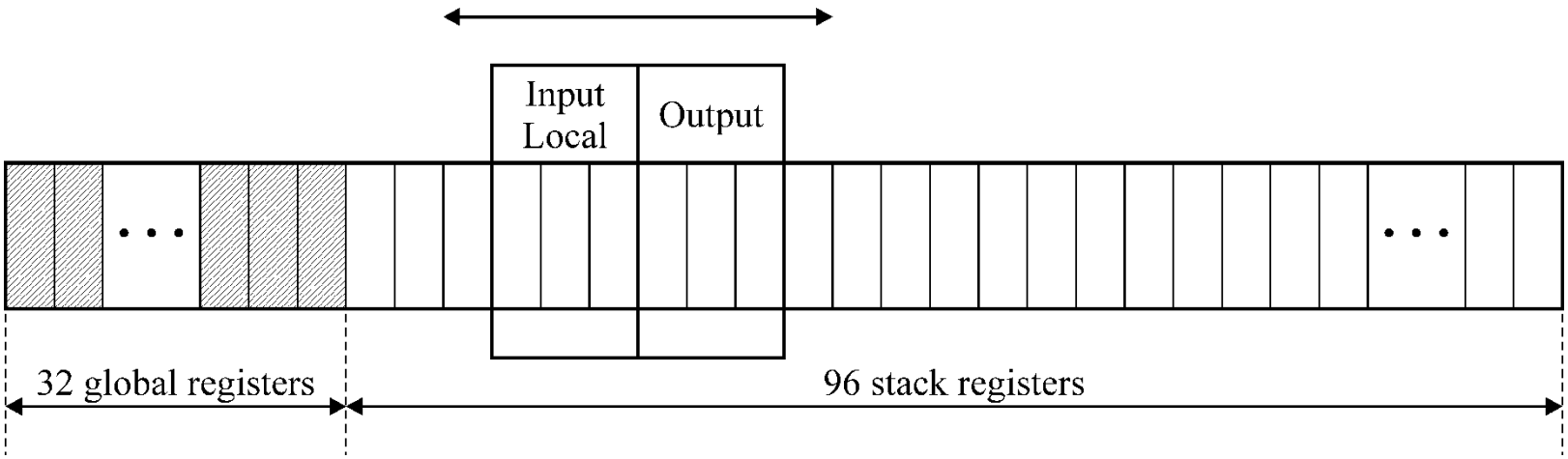
- Programowe potokowanie
  - Równoległe wykonanie różnych iteracji pętli
  - Wsparcie:
    - Automatyczne przemianowywanie rejestrów
      - Różne iteracje używają różnych podzbiorów rejestrów
    - Predykacja
      - Cykliczny rejestr predykatu określa fazę wykonania pętli
        - » prolog, jądro, epilog
    - Specjalna instrukcja końca pętli
      - Rotacja rejestrów
      - Dekrementacja licznika pętli

Intel Itanium

- Rejestry
  - 128×64-b stałoprzecinkowe
    - R0..R31 statyczne
    - R32..R127 przydzielane dynamicznie/cyklicznie
    - Bit NaT dla każdego rejestru
  - 128×80-b zmiennoprzecinkowe
    - Format rozszerzony zgodny z IEEE 754
    - R0..R31 statyczne
    - R32..R127 przydzielane dynamicznie/cyklicznie
  - 64×1-bs predykaty
    - Pr0=1 → rozkazy bezwarunkowe
    - Pr0..15 statyczne
    - pr16..63 przydzielane cyklicznie

Intel Itanium

- Stos rejestrów
  - Zapobiega zbędnym przesyłom rej↔pam podczas wywoływania podprogramów
  - Automatyczny przydział ramki
    - Do 96 rejestrów na procedurę



Intel Itanium

- Stos rejestrów
  - Wywołanie procedury
    - Ukrycie lokalnych rejestrów proc. wywołującej
    - Automatyczne przemianowywanie rejestrów
    - Przydział cykliczno-buforowy
      - Jeśli brak rejestrów, najstarsze przechowywane w pamięci