Wykonywanie instrukcji poza porządkiem programu

Maciej Buszka

Motywacja (względem prostego potoku)

- Generowany kod maszynowy abstrahuje od konkretnej implementacji procesora (pozwala na dystrybucję programów w postaci binarnej).
- Procesor może poradzić sobie z zależnościami nie do przewidzenia statycznie przez kompilator.
- Pozwala na “ukrywanie” operacji które mogą trwać długo, np. nietrafiony dostęp do pamięci podręcznej.
- Procesor maksymalizuje wykorzystanie jednostek funkcyjnych

Banalny program motywujący

1. DIV.D F0,F2,F4
2. ADD.D F10,F0,F8
3. SUB.D F12,F8,F14

- Zależność RAW między 2. a 1. linijką kodu
- Ale 3 instrukcja mogłaby zostać wykonana (brak zależności z poprzednimi)
- W przypadku potoku trzeba czekać
- Rozwiązanie: Wykonywanie instrukcji poza porządkiem programu.

Nowe hazardy

Przy przetwarzaniu poza porządkiem programu oprócz znanego nam hazardu

- RAW - read after write

możliwe są dwa nowe hazardy:

- WAR - write after read
- WAW - write after write

1. DIV.D F0,F2,F4
2. ADD.D F6,F0,F8
3. SUB.D F8,F10,F14
4. MUL.D F6,F10,F8

Wszystkie hazardy zostaną rozwiązane dzięki mechanizmowi przemianowania rejestrów

Wtrącenie o wyjątkach

- Widać, że wraz z porzuceniem szeregowego wykonywania instrukcji komplikuje się obsługa i zgłaszanie wyjątków.
- Procesor musi zadbać o to, by wyjątki były zaobserwowane w odpowiedniej kolejności.
- Możliwe, że wyjątki będą *nieprecyzyjne* tzn. stan procesora nie będzie taki sam jak przy wykonaniu szeregowym.
- Na razie nie będziemy się tym martwić, ponieważ rozwiązanie pojawia się “darmo” przy rozszerzaniu procesora o spekulację.

Algorytm Tomasulo

Idea

Dzielimy fazę wykonania na dwie części

- Zlecanie instrukcji
  - Wykonane sekwencyjnie
  - Sprawdzanie hazardów strukturalnych
- Wykonanie instrukcji
  - Dzieje się poza porządkiem programu (zlecania)
  - Instrukcje mogą się zakończyć w dowolnej kolejności
  - Wykorzystuje przemianowanie rejestrów do zapobieżenia hazardom
    - RAW
    - WAR
    - WAW
  - Wyjątki są nieprecyzyjne

Przemianowanie rejestrów

Przed:

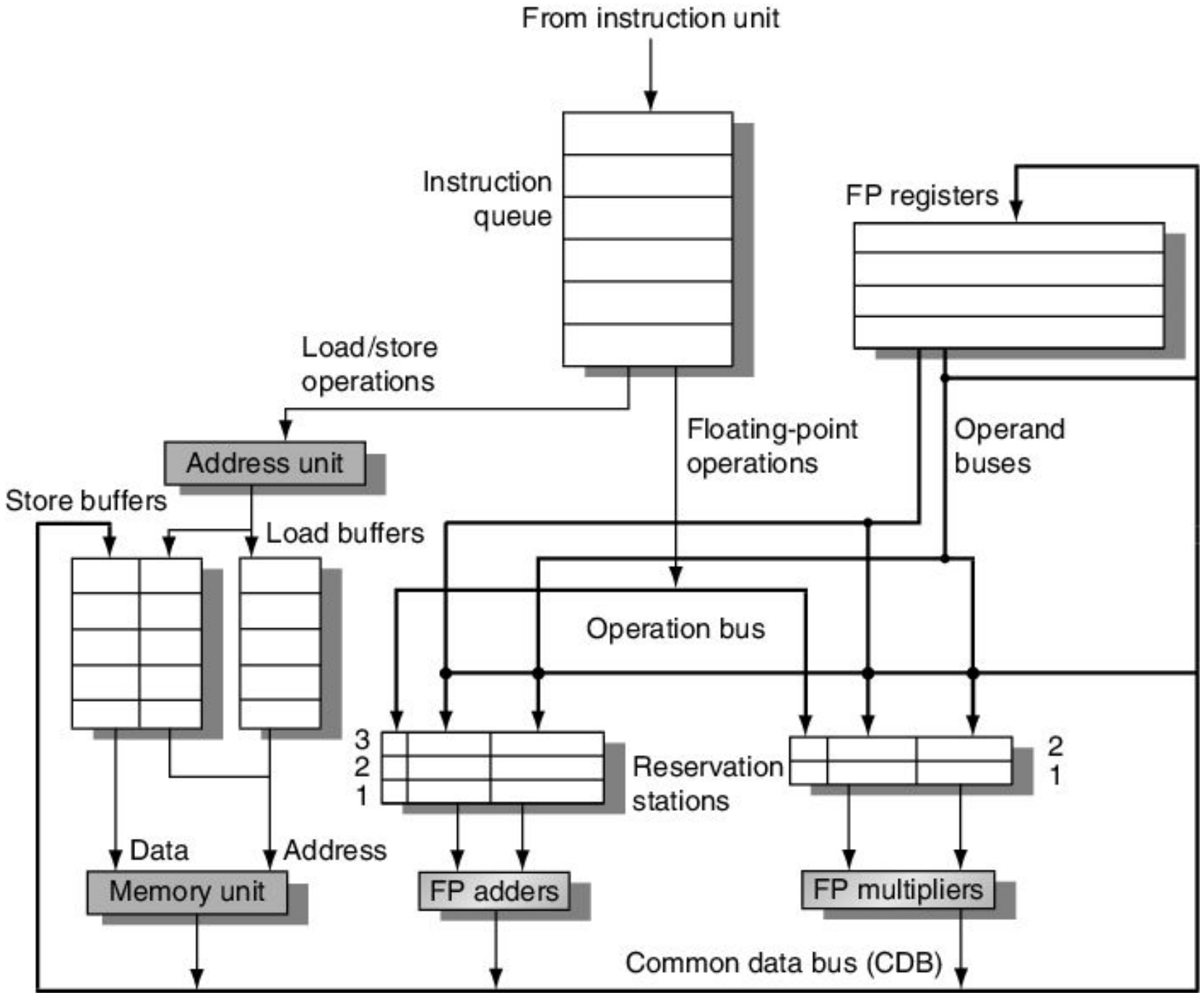
1. DIV.D F0,F2,F4
2. ADD.D F6,F0,F8
3. S.D F6,0(R1)
4. SUB.D F8,F10,F14
5. MUL.D F6,F10,F8

Po:

1. DIV.D F0,F2,F4
2. ADD.D T0,F0,F8
3. S.D T0,0(R1)
4. SUB.D T1,F10,F14
5. MUL.D F6,F10,T1

Kluczowe pojęcia

1. Przemianowanie rejestrów
   - Funkcja mapująca rejestr na numer stacji rezerwacji z której będzie pochodzić jego wartość.
   - Aktualizowana przy każdym zleceniu instrukcji.
2. Stacja rezerwacji
   - Buforuje instrukcję wraz z informacją o pochodzeniu jej argumentów
   - Zarazem służy jako identyfikator wyniku
3. Szyna danych
   - Wspólne połączenie między stacjami, rejestrami oraz jednostkami obliczeniowymi.
   - Po obliczeniu jakiejś instrukcji, jej wynik jest rozgłaszany na szynie (z informacją z której stacji pochodził) co pozwala na natychmiastowe zbuforowanie wyniku przez instrukcje na nią oczekujące, oraz być może zapisanie do rejestru
4. Bufory pamięci
   - Działają jak stacje rezerwacji dla dostępów do pamięci
   - Potrzebna jest dodatkowa logika sprawdzająca hazardy przez pamięć
   - Prosty sposób: adresy wyliczane w kolejności programu, nie można dodać nowej operacji wczytania dopóki istnieje oczekująca instrukcja zapisu pod ten sam adres. Zapisy muszą sprawdzać dodatkowo konfliktujące odczyty


From instruction unit
Instruction queue
FP registers
Load/store operations
Floating-point operations
Operand buses
Address unit
Store buffers
Load buffers
Operation bus
3
2
1
Reservation stations
2
1
Data
Address
Memory unit
FP adders
FP multipliers
Common data bus (CDB)

Przykład działania

1. L.D F6, 32(R2)
2. L.D F2, 44(R3)
3. MUL.D F0, F2, F4
4. SUB.D F8, F2, F6
5. DIV.D F10, F0, F6
6. ADD.D F6, F8, F2

Legenda:

- Vi - wartości operandów
- Qi - identyfikatory stacji z których będą pochodzić operandy

Stan wykonania instrukcji 1

Instruction		Instruction status		
		Issue	Execute	Write result
L.D	F6,32(R2)	√	√	√
L.D	F2,44(R3)	√	√	
MUL.D	F0,F2,F4	√		
SUB.D	F8,F2,F6	√		
DIV.D	F10,F0,F6	√		
ADD.D	F6,F8,F2	√		

	Reservation stations						
Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	No						
Load2	Yes	Load					44 + Regs[R3]
Add1	Yes	SUB		Mem[32 + Regs[R2]]	Load2		
Add2	Yes	ADD			Add1	Load2	
Add3	No						
Mult1	Yes	MUL		Regs[F4]	Load2		
Mult2	Yes	DIV		Mem[32 + Regs[R2]]	Mult1		

	Register status								
Field	F0	F2	F4	F6	F8	F10	F12	...	F30
Qi	Mult1	Load2		Add2	Add1	Mult2			

Stan wykonania instrukcji 2

Instruction		Instruction status		
		Issue	Execute	Write result
L.D	F6,32(R2)	√	√	√
L.D	F2,44(R3)	√	√	√
MUL.D	F0,F2,F4	√	√	
SUB.D	F8,F2,F6	√	√	√
DIV.D	F10,F0,F6	√		
ADD.D	F6,F8,F2	√	√	√

	Reservation stations						
Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	No						
Load2	No						
Add1	No						
Add2	No						
Add3	No						
Mult1	Yes	MUL	Mem[44 + Regs[R3]]	Regs[F4]			
Mult2	Yes	DIV		Mem[32 + Regs[R2]]	Mult1		

	Register status								
Field	F0	F2	F4	F6	F8	F10	F12	...	F30
Qi	Mult1					Mult2			

Rozszerzenie o spekulację

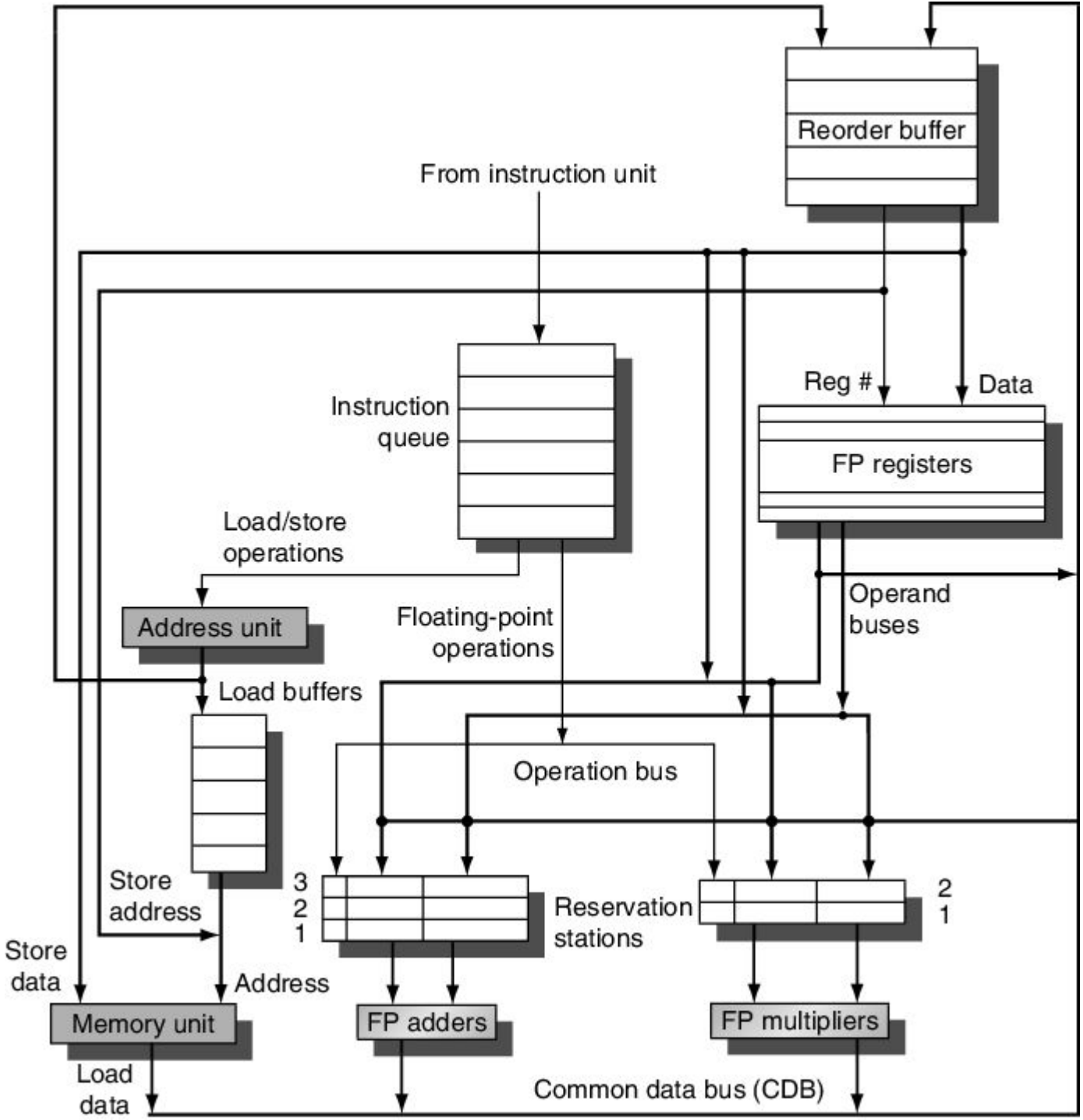
- Aby osiągnąć jeszcze lepsze wyniki chcemy przewidywać skoki i spekulatywnie wykonywać instrukcje
- Aby zapewnić odpowiednią semantykę, rozdzielamy wykonanie na 3 fazy:
  - Zlecenie
  - Wykonanie
  - Zatwierdzenie
- Ta ostatnia pozwala na przywrócenie odpowiedniej kolejności obserwacji efektów instrukcji
- Przy okazji załatwiamy wyjątki
  - Operacja może zwrócić wynik lub wyjątek
  - W momencie zatwierdzania podniesiemy ten wyjątek, jako, że mamy pewność iż ta instrukcja musiała zostać wykonana

Bufor przestawiania

- Każdy wpis odpowiada instrukcji w trakcie wykonania, bądź oczekującej na zatwierdzenie
- Instrukcje następujące po spekulacji mogą zostać zatwierdzone dopiero gdy zostanie ona potwierdzona
  - W przeciwnym wypadku usuwamy wszystkie wpisy dodane do bufora od spekulacji oraz zasilamy kolejkę instrukcji właściwymi instrukcjami
  - Procesor powinien jak najszybciej sprawdzać poprawność spekulacji by uniknąć długich przestojów
- Nowo zlecane instrukcje są zasilane wartościami z bufora, lub zapamiętują z którego wpisu powinny pobrać wartość gdy zostanie ona wyprodukowana (jednym słowem bufor służy za dodatkowy zestaw rejestrów)

Bufor przestawiania a pamięć

- Służy jako kolejka zapisów
  - Zapis następuje w momencie zatwierdzania instrukcji
- Odczyty są buforowane jak wcześniej
  - Znów trzeba zadbać o brak hazardów przez pamięć
  - WAW oraz WAR nam nie grożą, jako że zapisy następują w kolejności programu
  - RAW załatwiamy za pomocą dwóch reguł
    - Odczyt nie może zostać wykonany jeśli w buforze przestawiania oczekuje zapis do tego samego adresu
    - Obliczanie adresu odczytu jest utrzymywane w kolejności programu względem wcześniejszych zapisów (czyli wiemy kiedy będzie konflikt)


Reorder buffer
From instruction unit
Instruction queue
Reg #
Data
FP registers
Load/store operations
Address unit
Floating-point operations
Operand buses
Load buffers
Operation bus
Store address
3
2
1
Reservation stations
2
1
Store data
Address
Memory unit
FP adders
FP multipliers
Load data
Common data bus (CDB)

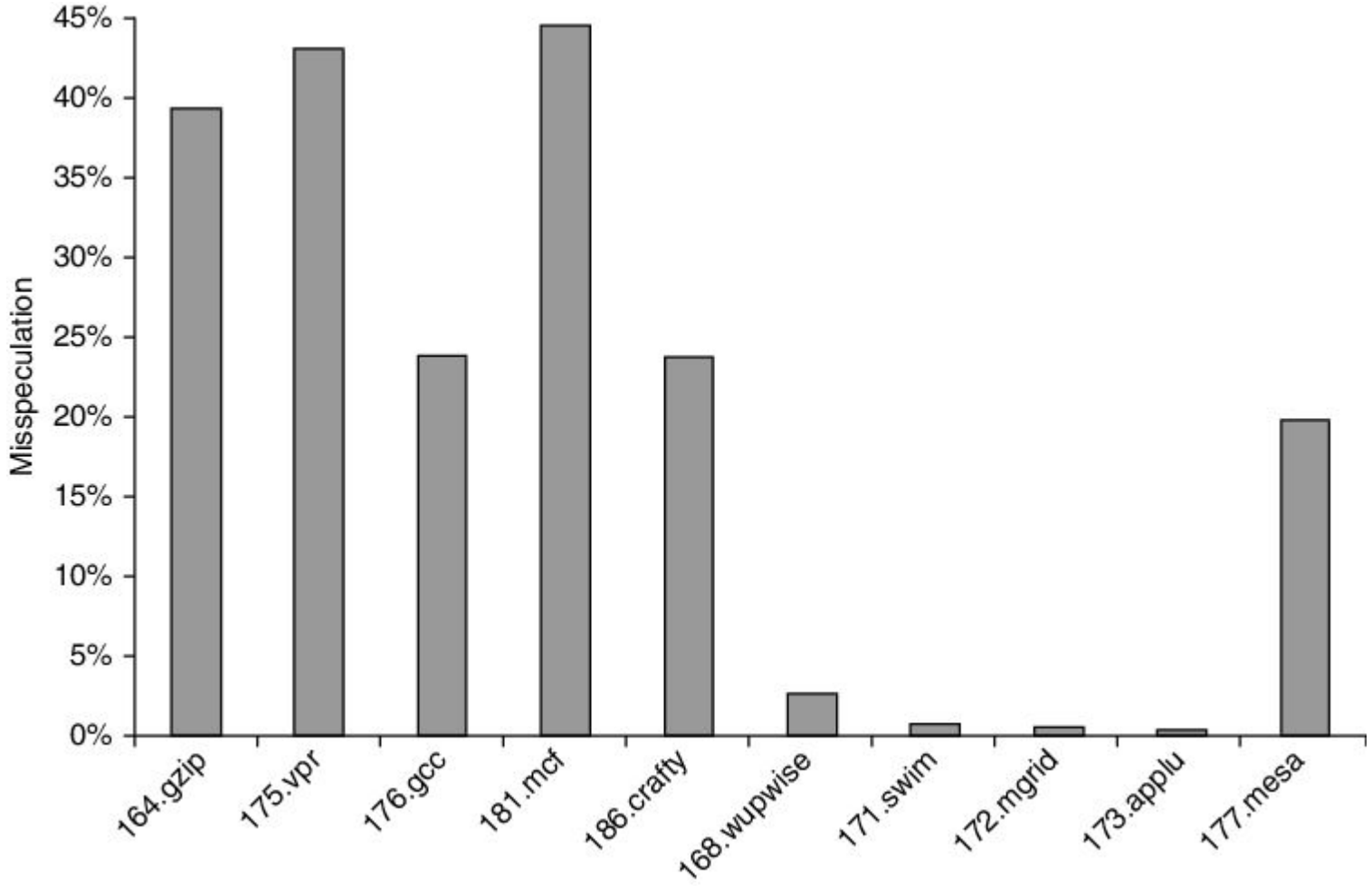
Przykład działania

```
1.  Loop: L.D      F0,0(R1)
2.        MUL.D    F4,F0,F2
3.        S.D      F4,0(R1)
4.        DADDIU   R1,R1,#-8
5.        BNE      R1,R2,Loop   ; branches if R1 != R2
```

	Reorder buffer				
Entry	Busy	Instruction	State	Destination	Value
1	No	L.D F0,0(R1)	Commit	F0	Mem[0 + Regs[R1]]
2	No	MUL.D F4,F0,F2	Commit	F4	#1 × Regs[F2]
3	Yes	S.D F4,0(R1)	Write result	0 + Regs[R1]	#2
4	Yes	DADDIU R1,R1,#-8	Write result	R1	Regs[R1] – 8
5	Yes	BNE R1,R2,Loop	Write result		
6	Yes	L.D F0,0(R1)	Write result	F0	Mem[#4]
7	Yes	MUL.D F4,F0,F2	Write result	F4	#6 × Regs[F2]
8	Yes	S.D F4,0(R1)	Write result	0 + #4	#7
9	Yes	DADDIU R1,R1,#-8	Write result	R1	#4 – 8
10	Yes	BNE R1,R2,Loop	Write result		

	FP register status								
Field	F0	F1	F2	F3	F4	F5	F6	F7	F8
Reorder #	6				7				
Busy	Yes	No	No	No	Yes	No	No	...	No

Koszty energetyczne spekulacji



Kilka haseł

- Równoczesne wykonywanie instrukcji trwających wiele cykli
- Równoczesne wykonywanie wielu iteracji tej samej pętli
- Ukrywanie wolnych dostępów do pamięci
- Gwarancja poprawności obserwowanych efektów oraz wyjątków
- Spekulację można zaobserwować ze względu na cache
- Często procesor nie będzie spekulatywnie wykonywać dostępu do L3 cache, tudzież rozwiązywał TLB miss, jako że jest to zbyt kosztowne

Co dalej?

Zlecanie i zatwierdzanie wielu instrukcji na cykl

- Dwa podejścia
- Zlecanie co pół cyklu
  - Proste rozwiązanie
  - Niestety nie skaluje się dalej
- Zlecanie równolegle
  - Trzeba rozstrzygać zależności między instrukcjami w jednej paczce
  - Ich liczba rośnie kwadratowo z wielkością paczki!
  - Wszystko to musi się dziać w jednym cyklu

Równoczesne wykonywanie wielu wątków

Proste rozszerzenie modelu procesora ze spekulacją:

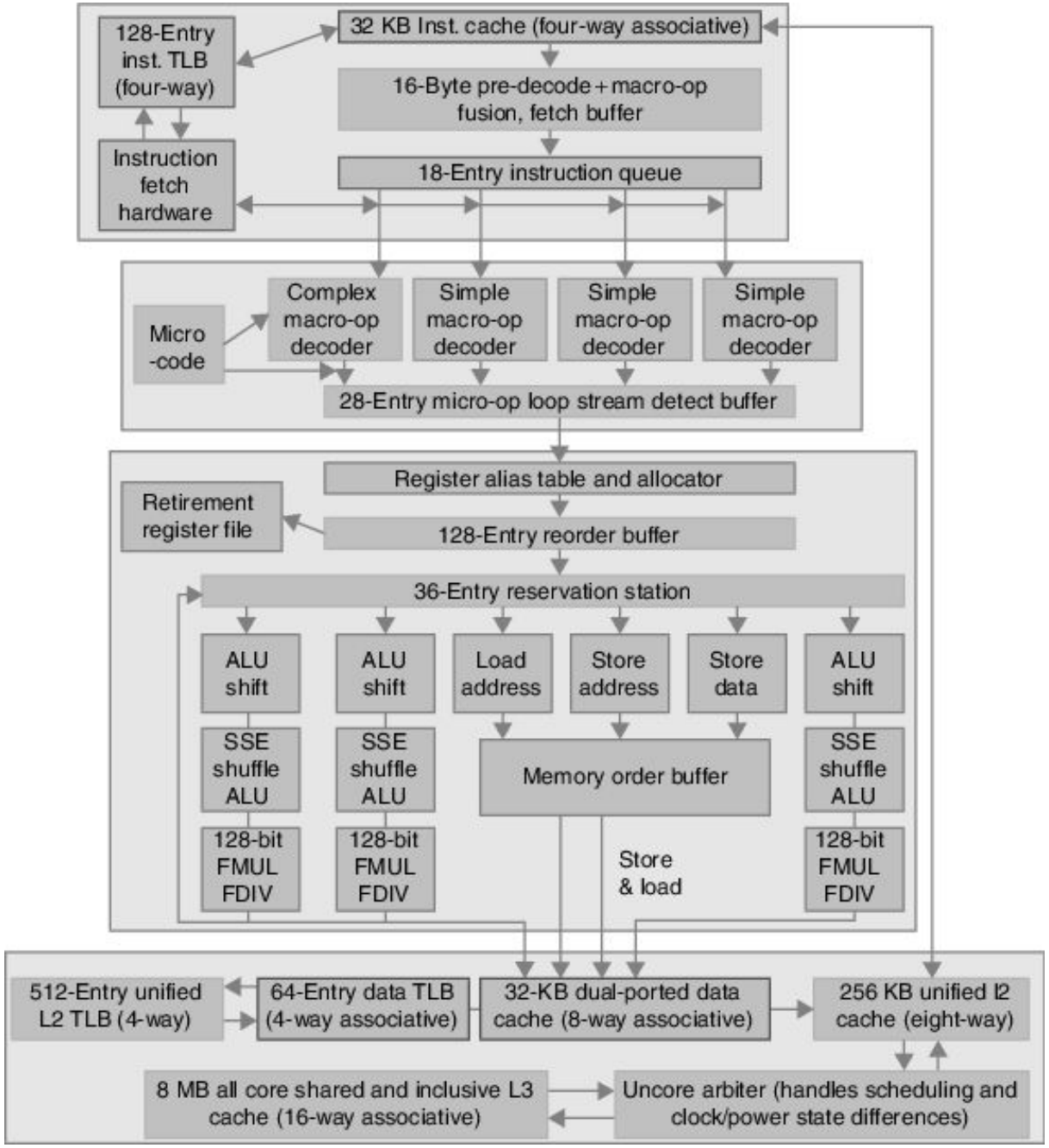
- Dodatkowa kolejka instrukcji i jednostka pobierająca
- Dodatkowy zestaw rejestrów

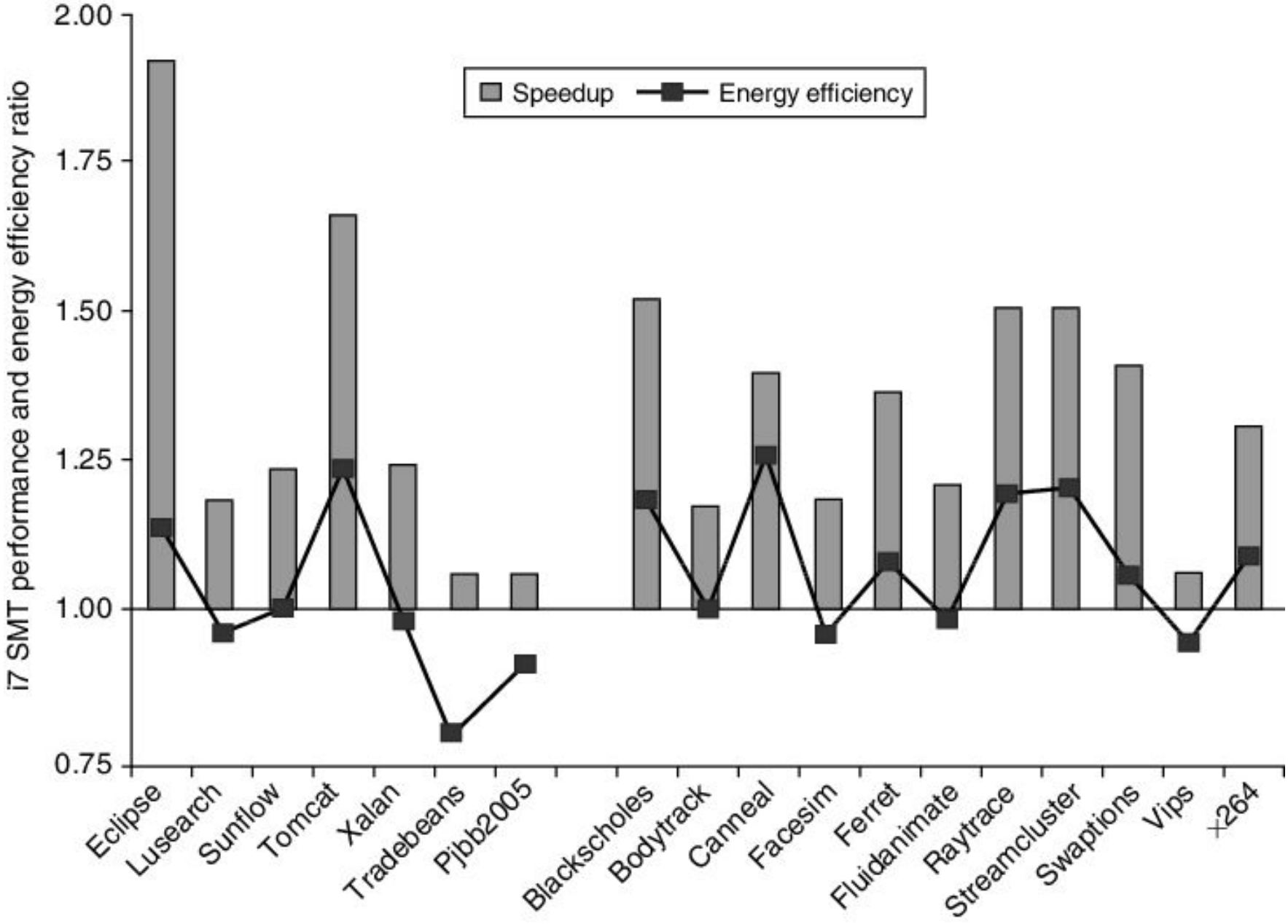
Intel nazywa to HyperThreading

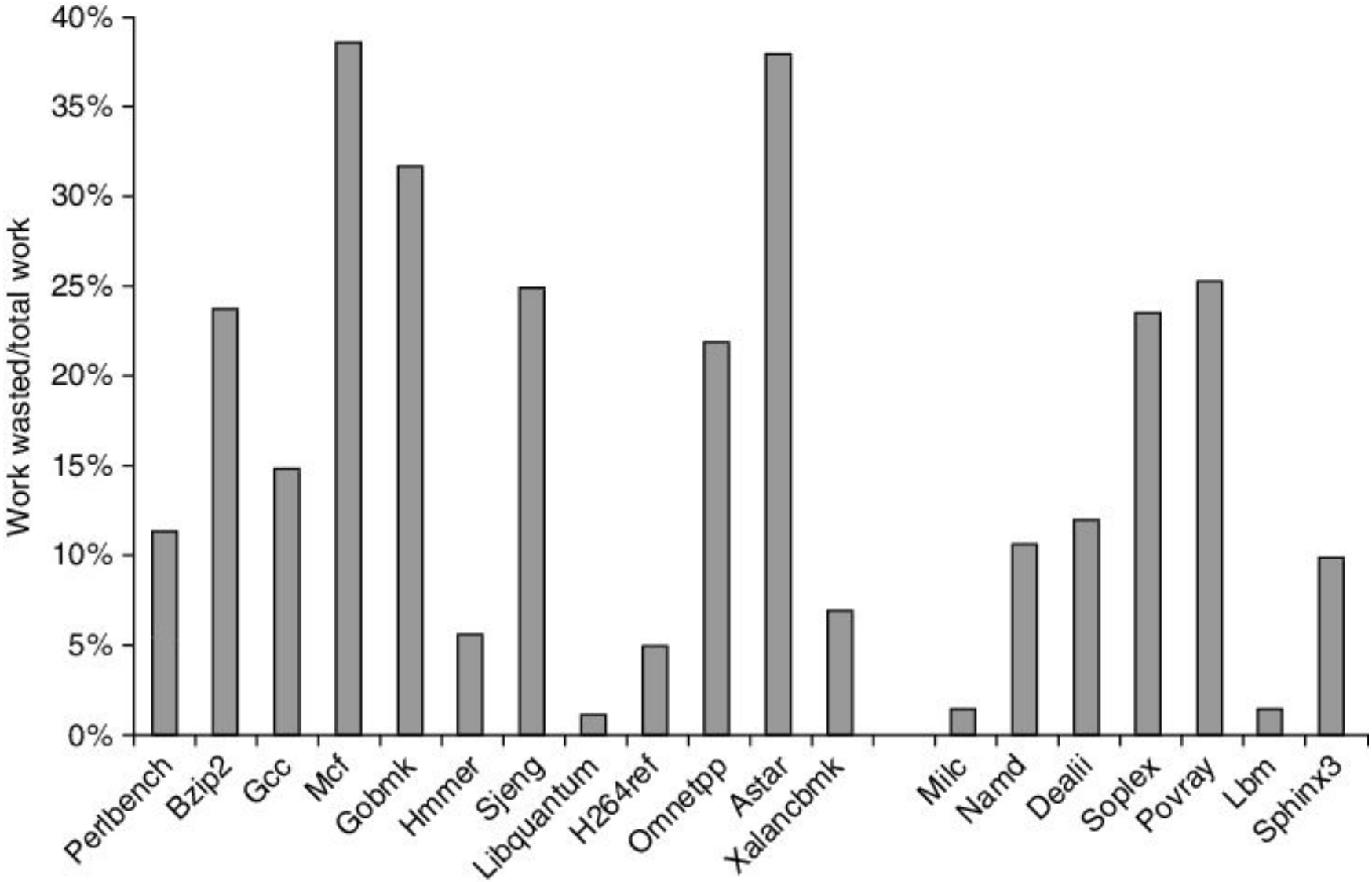
Przykład współczesnego procesora Core i7

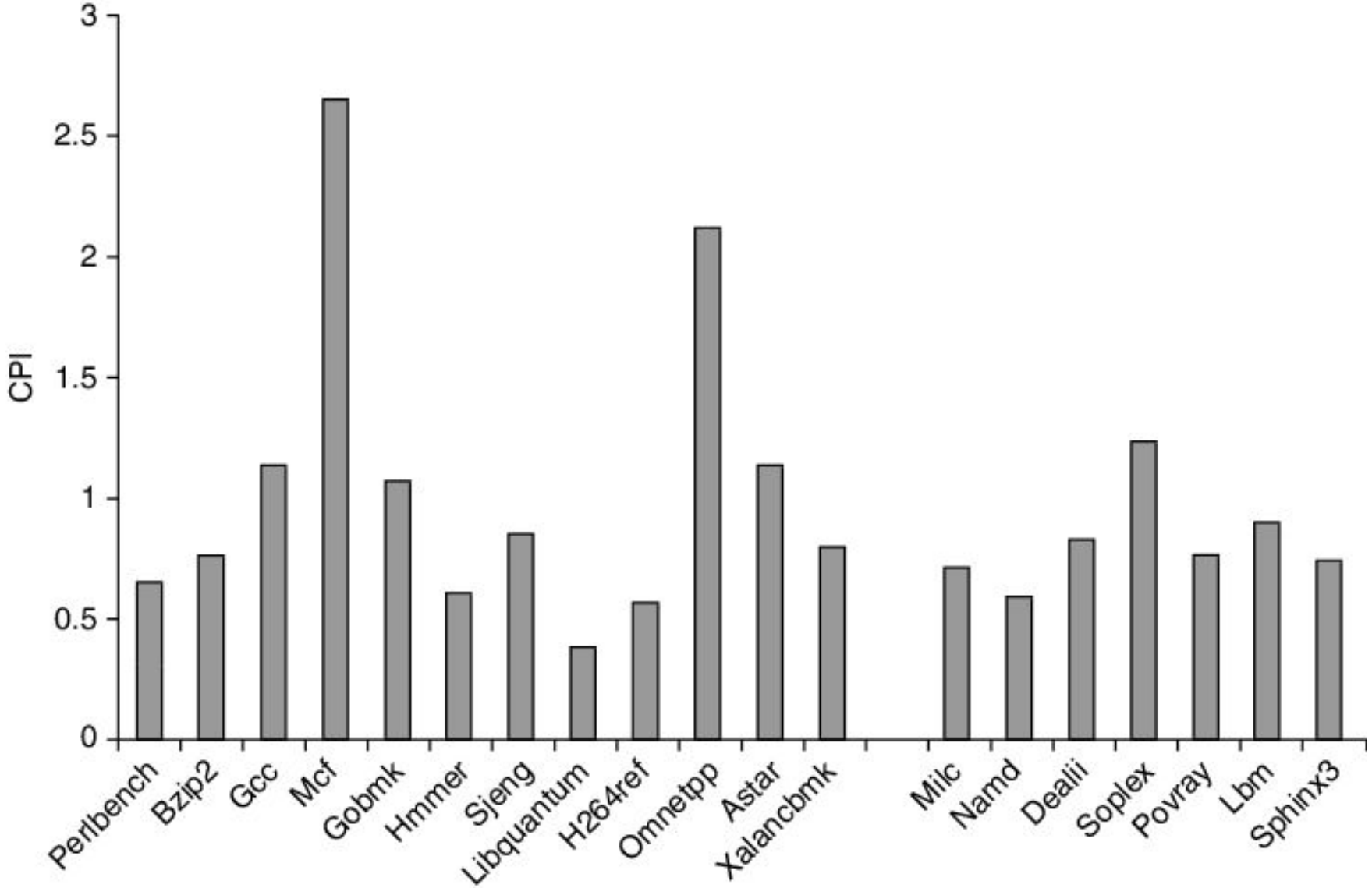
Na następnych slajdach:

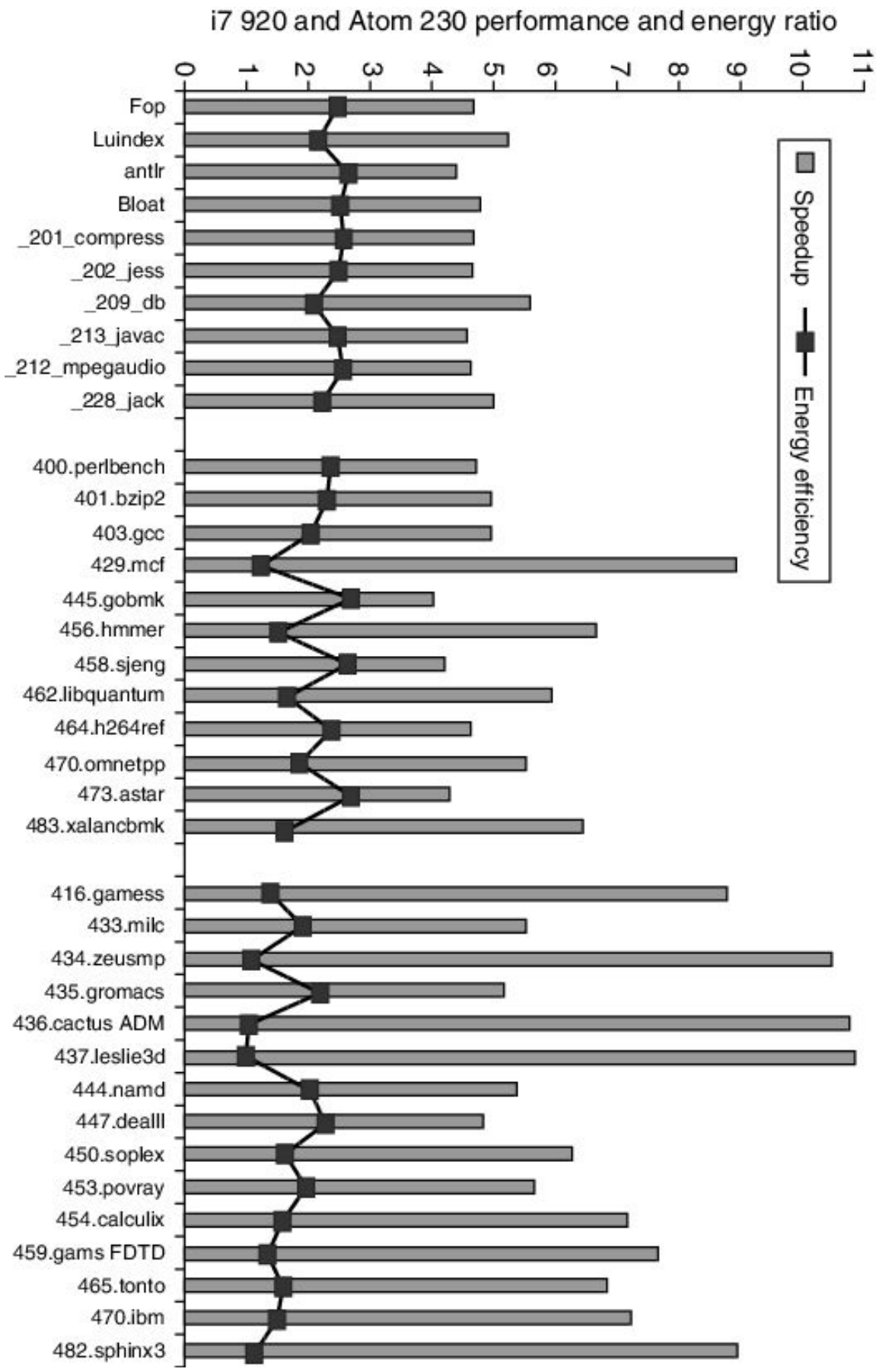
1. Pobieżny schemat procesora
2. Zyski wydajności ze względu na równoczesne wykonywanie kilku wątków na jednym rdzeniu
3. Stosunek instrukcji zatwierdzonych do wykonanych (zmarnowana praca)
4. Średnia ilości cykli na instrukcję
5. i7 vs Atom (benchmarki jednowątkowe)
    a. Atom wykonuje instrukcje w porządku programu
    b. Około 2 razy wolniejszy zegar


128-Entry inst. TLB (four-way)
32 KB Inst. cache (four-way associative)
16-Byte pre-decode + macro-op fusion, fetch buffer
Instruction fetch hardware
18-Entry instruction queue
Micro -code
Complex macro-op decoder
Simple macro-op decoder
Simple macro-op decoder
Simple macro-op decoder
28-Entry micro-op loop stream detect buffer
Register alias table and allocator
Retirement register file
128-Entry reorder buffer
36-Entry reservation station
ALU shift
ALU shift
Load address
Store address
Store data
ALU shift
SSE shuffle ALU
SSE shuffle ALU
Memory order buffer
SSE shuffle ALU
128-bit FMUL FDIV
128-bit FMUL FDIV
Store & load
128-bit FMUL FDIV
512-Entry unified L2 TLB (4-way)
64-Entry data TLB (4-way associative)
32-KB dual-ported data cache (8-way associative)
256 KB unified l2 cache (eight-way)
8 MB all core shared and inclusive L3 cache (16-way associative)
Uncore arbiter (handles scheduling and clock/power state differences)


i7 SMT performance and energy efficiency ratio
2.00
1.75
1.50
1.25
1.00
0.75
Speedup
Energy efficiency
Eclipse
Lusearch
Sunflow
Tomcat
Xalan
Tradebeans
Pjbb2005
Blackscholes
Bodytrack
Canneal
Facesim
Ferret
Fluidanimate
Raytrace
Streamcluster
Swaptions
Vips
+264


Work wasted/total work
40%
35%
30%
25%
20%
15%
10%
5%
0%
Perlbench
Bzip2
Gcc
Mcf
Gobmk
Hmmer
Sjeng
Libquantum
H264ref
Omnetpp
Astar
Xalancbmk
Milc
Namd
Dealii
Soplex
Povray
Lbm
Sphinx3


CPI
3
2.5
2
1.5
1
0.5
0
Perlbench
Bzip2
Gcc
Mcf
Gobmk
Hmmer
Sjeng
Libquantum
H264ref
Omnetpp
Astar
Xalancbmk
Milc
Namd
Dealii
Soplex
Povray
Lbm
Sphinx3


i7 920 and Atom 230 performance and energy ratio
0
1
2
3
4
5
6
7
8
9
10
11
Speedup
Energy efficiency
Fop
Luindex
antlr
Bloat
_201_compress
_202_jess
_209_db
_213_javac
_212_mpegaudio
_228_jack
400.perlbench
401.bzip2
403.gcc
429.mcf
445.gobmk
456.hmmer
458.sjeng
462.libquantum
464.h264ref
470.omnetpp
473.astar
483.xalancbmk
416.gamess
433.milc
434.zeusmp
435.gromacs
436.cactus ADM
437.leslie3d
444.namd
447.dealII
450.soplex
453.povray
454.calculix
459.gams FDTD
465.tonto
470.ibm
482.sphinx3

Literatura

- Hannessy, Patterson: Computer Architecture
- Shen, Lipasti: Modern Processor Design