Architektury systemów komputerowych 2017

Lista zadań nr 14

Na zajęcia 14–16 czerwca 2017

UWAGA! W trakcie prezentacji zadań należy być przygotowanym do wytłumaczenia haseł, które zostały oznaczone **wytłuszczoną** czcionką.

W zadaniach odnoszących się do potokowej realizacji procesora MIPS zakładamy, że procesor nie implementuje *branch delay slots* chyba, że podano inaczej.

Zadanie 1. Rozważamy procesor potokowy MIPS, którego schemat pojawił się na poprzedniej liście zadań. Wiemy, że skoki warunkowe wykonują się na początku fazy EX i nie ma implementacji *branch delay slots*.

```
1 l1:
2     addi  $3,$5,4
3     lw    $4,0($6)
4     beq   $2,$3,l2  # (a) skok się wykonał (b) skok się nie wykonał
5     addi  $2,$2,4
6 l2:
7     bne   $2,$4,l1  # (a) skok się nie wykonał (b) skok się wykonał
8     sw    $3,0($2)
9     addi  $3,$2,$3
```

Przyjmujemy strategię **statycznego przewidywania skoków** „*zawsze wykonaj skok*". Dla obydwu wariantów wykonania powyższego ciągu instrukcji narysuj diagram stanu potoku (jak na slajdzie 61).

Zadanie 2. Rozważmy potokowy procesor MIPS z implementacją obejść i wykrywania hazardów, w którym skoki wykonują się na początku fazy EX. Ile cykli zajmuje wykonanie jednej iteracji[1] poniższej pętli?

```
1 loop:
2     lw    $t0,0($a0)    # A[i]
3     shl   $t0,$t0,7     # A[i] << 7
4     addu  $t1,$t0,$v0   # (A[i] << 7) + a
5     lw    $t0,0($a1)    # B[i]
6     shr   $t0,$t0,9     # B[i] >> 9
7     addu  $t0,$t0,$v1   # (B[i] >> 9) + b
8     xor   $t1,$t0,$t1   # ((A[i] << 7) + a) ^ ((B[i] >> 9) + b)
9     sw    $t1,0($a2)    # C[i] = ((A[i] << 7) + a) ^ ((B[i] >> 9) + b)
10    addi  $a0,$a0,4
11    addi  $a1,$a1,4
12    addi  $a2,$a2,4
13    bne   $a2,$a3,loop  # $a3 = koniec tablicy C
```

Wykonaj **optymalizację**[2] kodu powyższej pętli. Można korzystać z dodatkowych rejestrów $t2...t7. Ile cykli zajmuje wykonanie jednej iteracji pętli po optymalizacji?

Zadanie 3. Odpowiedz na pytania z poprzedniego zadania dla superskalarnego procesora MIPS z dwoma potokami: U-pipe, który wykonuje wszystkie instrukcje, oraz V-pipe, który wykonuje wyłącznie instrukcje arytmetyczno-logiczne i skoki. Analizując wykonanie kodu posłuż się tabelką ze slajdu nr 118.

[1] Chodzi o czas od momentu wejścia pierwszej instrukcji do etapu IF do wyjścia ostatniej instrukcji z etapu WB.

[2] Dopuszczamy dowolną transformację kodu zachowującą semantykę, np. zamiana kolejnością lub zastępowanie instrukcji.

Zadanie 4. Rozważamy potokowy procesor MIPS z implementacją instrukcji wielocyklowych. Zakładamy, że procesor implementuje obejścia, a skoki wykonują się na początku fazy EX. Przyjmujemy następujące wartości **czasu przetwarzania instrukcji**: EX – 1, MEM – 2, FP-ADD – 2, FP-MUL – 6, FP-DIV – 12. **Interwał inicjacji** instrukcji wynosi 1 z wyjątkiem FP-MUL – 2 i FP-DIV – 11. Ile cykli zajmie wykonanie jednej iteracji przed i po optymalizacji?

```
1 loop:
2     ldc1   $f2,0($a1)   # X[i]
3     mul.d  $f4,$f2,$f0  # a * X[i]
4     ldc1   $f6,0($a2)   # Y[i]
5     add.d  $f6,$f4,$f6  # a * X[i] + Y[i]
6     sdc1   $f6,0($a2)   # Y[i] = a * X[i] + Y[i]
7     addi   $a1,$a1,8
8     addi   $a2,$a2,8
9     bne    $a2,$a3,loop # $a3 = koniec tablicy Y
```

Zadanie 5. Powróćmy do kodu z poprzedniego zadania i użyjemy **rozwijania pętli** celem dalszej minimalizacji czasu przetwarzania elementów tablicy Y. Rozwiń pętlę tak, by jeden przebieg pętli obliczał dwa elementy. Zoptymalizuj kod i oblicz czas przetwarzania jednego elementu. Czy dalsze rozwijanie pętli przyniesie zwiększenie wydajności kodu?

Na potrzeby poniższych zadań, w razie wątpliwości, można odnieść się do rozdziału 2.5 książki „*Computer Architecture: A Quantitative Approach*", gdzie opisane są szczegóły algorytmu Tomasulo.

Zadanie 6. Uruchamiamy kod pętli z drugiego zadania na procesorze wykonującym instrukcje **poza porządkiem programu** (ang.*out-of-order execution*). Wskaż hazardy danych typu **Write-after-Write** i **Write-after-Read**. Wyjaśnij jak mikroarchitektura *Out-of-Order* wykorzystuje technikę **przemianowywania rejestrów** do eliminacji powyższych hazardów. Podaj zawartość bufora przemianowywania rejestrów[3] po zleceniu każdej instrukcji operującej na rejestrach $t0 i $t1.

Zadanie 7. Na przykładzie kodu z pierwszego zadania wytłumacz jak procesor *Out-of-Order* ukrywa opóźnienia związane z dostępem do pamięci. Dla instrukcji lw i sw czas przetwarzania wynosi 4 cykle, a interwał inicjacji 2 cykle. Ile czasu[4] zajmie przetworzenie pierwszych dwóch iteracji pętli na idealnej mikroarchitekturze *Out-of-Order* – tj. nie ma ograniczeń na ilość zasobów sprzętowych.

Zadanie 8 (bonus, 2pkt). Rozważmy procesor *Out-of-Order* potrafiący **zlecić** i **zatwierdzić** do dwóch instrukcji w jednym cyklu. Dysponujemy czterema **jednostkami funkcyjnymi**, które przetwarzają: dostępy do pamięci (LSU), skoki (BPU) i operacje arytmetyczno-logiczne (ALU0 i ALU1). Wszystkie **poczekalnie** (ang. *reservation station*) są jednoelementowe. **Jednostka zatwierdzania** przechowuje do 8 instrukcji. **Bufor przemianowania rejestrów** ma miejsce na 4 wpisy. **Kolejka instrukcji** jest ładowana przez wyrocznię (predyktor skoków ze 100% skuteczności) i zawsze przechowuje 4 instrukcje z właściwej ścieżki programu.

Przedstaw symulację wykonania pierwszych dwóch iteracji pętli z zadania pierwszego. Instrukcjom przypisz numery odpowiadające pozycjom w jednostce zatwierdzania. W każdym cyklu wskaż stan poczekalni, bufora przemianowania, jednostek funkcyjnych i jednostki zatwierdzania.

[3] W uproszczeniu – asocjacyjna pamięć mapująca numer rejestru na znacznik instrukcji, która obliczy jego wartość.
[4] Tym razem chodzi o ilość cykli od zlecenia pierwszej instrukcji, do zatwierdzenia ostatniej.