Architektury systemów komputerowych 2016

Lista zadań nr 13 (szczęśliwa)

Na zajęcia 6–9 czerwca 2016

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

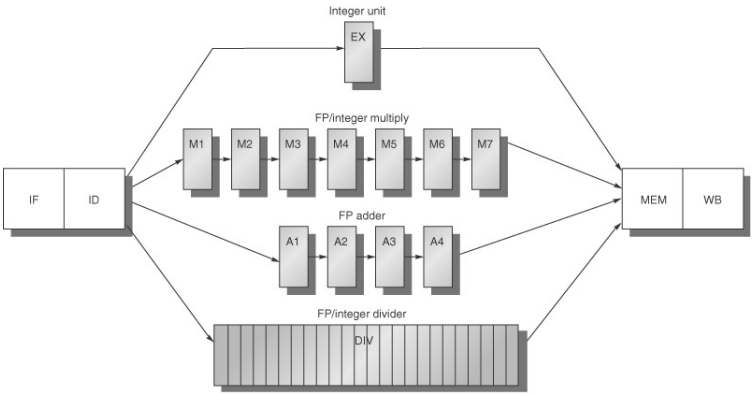
Zadanie 1. Rozważmy potokowy procesor MIPS z implementacją obejść i wykrywania hazardów, w którym skoki wykonują się w fazie EX. Ile cykli zajmuje wykonanie jednej iteracji[1] poniższej pętli?

```
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 2. Powtórz polecenia 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. Analizując wykonanie kodu posłuż się tabelką ze slajdu nr 118.

Zadanie 3. Poniżej przedstawiono potokowy procesor MIPS z implementacją instrukcji wielocyklowych. Zakładamy, że procesor implementuje obejścia, a skoki wykonują się w fazie EX.



Przyjmujemy następujące wartości **czasu przetwarzania instrukcji**: EX – 1, MEM – 2, FP-ADD – 3, FP-MUL – 6, FP-DIV – 12. **Interwał inicjacji** instrukcji wynosi 1 z wyjątkiem FP-DIV – 11. Ile taktów zegara zajmie jednokrotne wykonanie iteracji poniżej pętli?

[1] Chodzi o czas od momentu wejścia pierwszej instrukcji do etapu IF do wyjścia ostatniej instrukcji z etapu WB.
[2] Można dodawać instrukcje, usuwać, modyfikować i zamieniać kolejnością.

```
1  loop:
2      ldc1   $f0,0($a0)    # X[i]
3      mul.d  $f2,$f8,$f0   # a * X[i]
4      ldc1   $f4,0($a1)    # Y[i]
5      mul.d  $f6,$f10,$f4  # b * Y[i]
6      add.d  $f0,$f2,$f6   # a * X[i] + b * Y[i]
7      div.d  $f2,$f0,$f12  # (a * X[i] + b * Y[i]) / c
8      sdc1   $f2,0($a2)    # Z[i] = (a * X[i] + b * Y[i]) / c
9      addi   $a0,$a0,8
10     addi   $a1,$a1,8
11     addi   $a2,$a2,8
12     bne    $a2,$a3,loop  # $a3 = koniec tablicy Y
```

Ile cykli zajmuje wykonanie jednej iteracji powyższej pętli po optymalizacji?

Zadanie 4. Rozważmy wykonanie poniższej pętli na procesorze z poprzedniego zadania. Ile cykli zajmuje 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
```

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 5. Uruchamiamy kod pętli z pierwszego 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 6. 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 7 (2). 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 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.