Architektury systemów komputerowych

Ćwiczenia 6: "Zaawansowana mikroarchitektura"

Należy przygotować się do zajęć czytając następujące rozdziały książek:

- Computer Architecture: A Quantitative Approach (4 edycja)

Należy być przygotowanym do wytłumaczenia **wytłuszczonych** haseł.

Zadanie 1 [§2.2]

Przypomnij definicję hazardu **strukturalnego**, **danych** i **kontroli**. Zdefiniuj hazardy **RAW**, **WAR**, **WAW** i na przykładzie kawałków kodu wytłumacz w jakich sytuacjach mogą występować zakładając, że wykonujemy instrukcje na procesorze o **mikroarchitekturze out-of-order**.

Zadanie 2 [§A.5]

Na potokowym procesorze MIPS (rysunek A.31) z wielocyklową jednostką FPU wykonujemy poniższą pętlę:

```
Loop: l.d         $f2,0($a1)        # $a1 = &X[i]
      add.d       $f2,$f0,$f2       # X[i] + a
      div.d       $f8,$f2,$f0       # (X[i] + a) / b
      l.d         $f4,0($a2)        # $a2 = &Y[i]
      mul.d       $f4,$f0,$f4       # Y[i] * a
      add.d       $f10,$f8,$f2      # (X[i] + a) / b + Y[i] * a
      s.d         $f4,0($a2)
      addi        $a0,$a0,8
      addi        $a1,$a1,8
      slt         $t0,$a2,$a3       # $a3 = koniec tablicy Y (1024 elementy)
      bnez        $t0,Loop
```

Wyjaśnij czym jest **opóźnienie instrukcji** i **interwał inicjacji**. Opóźnienia instrukcji są nastepujące – ALU: 0, MEM: 1, FP ADD: 3, FP/INT MUL: 6, FP/INT DIV: 12; a interwał inicjacji wynosi 1 z wyjątkiem FP/INT DIV: 11. Ile cykli zegarowych zajmie pojedyncze wykonanie pętli licząc od momentu wyjścia pierwszej instrukcji z etapu WB do wyjścia ostatniej instrukcji z etapu WB? Czy w trakcie wykonania kodu występują hazardy WAR i WAW?

Zadanie 3 [§A.5,§2.2]

Zoptymalizuj poniższą petlę do wykonania na procesorze opisanym w poprzednim zadaniu. Możesz zmieniać kolejność instrukcji, nazwy rejestrów i przeprowadzać inne **transformacje kodu** przy zachowaniu semantyki. Ile iteracji pętli opłaca się rozwinąć, aby istotnie zmniejszyć ilość cykli per element tablicy (z drugiej strony chcemy, żeby kod wynikowy był jak najkrótszy).

```
Loop: l.d         $f2,0($a1)        # $a1 = &X[i]
      mul.d       $f4,$f2,$f0       # $f0 = a
      l.d         $f6,0($a2)        # $a2 = &Y[i]
      add.d       $f6,$f4,$f6
      s.d         $f6,0($a2)        # Y[i] = a * X[i] + Y[i]
      addi        $a1,$a1,8
      addi        $a2,$a2,8
      slt         $t0,$a2,$a3       # $a3 = koniec tablicy Y (1024 elementy)
      bnez        $t0,Loop
```

Zadanie 4 [§A.7]

Na podstawie kodu z zadania 2 zaprezentuj działanie procesora MIPS z implementacją **tablicy wyników** (ang. *scoreboard*). Jak będzie wyglądał stan tablicy (tabela A.51) w momencie, gdy instrukcja `slt` osiągnie etap zapisu rezultatu? Załóż, że procesor ma strukturę jak na rysunku A.50 z dodatkiem jednej jednostki *Load-Store*. która obsługuje operacje na pamięci. Opóźnienia instrukcji i interwał inicjacji mają wartości jak w zadaniu 2. Czy przy wykonaniu kodu występują hazardy danych WAR i WAW?

Zadanie 5 [§2.2]

Przetłumacz poniższy kod w języku C na asembler MIPS przy założeniu, że kod będzie wykonywany na idealnym superskalarnym procesorze ze **statycznym planowaniem** instrukcji (ang. *static scheduling*) i **podwójnym zlecaniem instrukcji** (ang. *dual issue*). Policz ile cykli zajmie wykonanie iteracji. Następnie skorzystaj z faktu, że N jest parzyste i rozwiń pętlę. Jaki jest zysk z takiej optymalizacji – jak zmienił się czas wykonania iteracji?

```
for (i = 0; i < N; i++)
  x[i] = 2 * x[i] + 5;
```

Zadanie 6 [§2.3]

Dwubitowy licznik nasycający jest lokalnym predyktorem, który potrafi dość dobrze opisywać zachowanie skoków bez uwzględnienia kontekstu. Niestety okazuje się, że w programach kolejne instrukcje skoków są często od siebie zależne i można używać tej własności do konstrukcji lepszego tzw. korelującego predyktora skoków. Zaprezentuj prosty kod dla którego dwupoziomowy adaptywny predyktor (Yale Patt) będzie odznaczał się większą skutecznością niż dwubitowy licznik nasycający. Z pomocą rysunku przedstaw strukturę predyktora (4,2) z tabelą historii o 4096 wpisach i wyjaśnij w jaki sposób dla danego licznika programu predyktor odpowiada na pytanie czy skok się wykona czy nie.

Zadanie 7 [§2.4, §2.5]

W zadaniu 4 zapoznaliśmy się z mikroarchitekturą, która potrafiła **wykonywać instrukcje poza porządkiem**. Algorytm **tablicy wyników** nie umożliwiał **zlecania instrukcji poza porządkiem** (ang. *out-of-order issue*) ani **zlecania wielu instrukcji** w jednym cyklu (ang. *wide issue*). Na podstawie kodu MIPS wytłumacz jak technika nazywana przemianowywaniem rejestrów (ang. *register renaming*) pozwala unikać hazardów WAR i WAW. Z użyciem rysunku 2.9 wytłumacz działanie **algorytmu Tomasulo**. Jak są rozwiązywane hazardy strukturalne i hazardy danych? W jaki sposób można rozpocząć wykonanie więcej niż jednej instrukcji w jednym cyklu zegarowym? Jakie problemy sprawiają instrukcje operujące na pamięci?

Zadanie 8 [§2.5]

Wykonujemy poniższy kod na procesorze o architekturze z poprzedniego zadania. Nie musimy się martwić o instrukcje skoku – zakładamy idealny predyktor skoków, zatem instrukcje skoków nie pojawiaja się w kolejce instrukcji. Zakładamy również, że instrukcje *Load-Store* mają 3 cyklowe opóźnienie, a instrukcje *ALU* 1 cyklowe. Przeprowadź dwie pierwsze iteracje pętli w podobny sposób jak zostało to przedstawione w tabeli 2.13.

```
        # i: $s0 = 0
        # n: $s1 = 1024
        # c: $s2 = 0
Loop:   add   $t0,$s2,$s0       # c + i
        sw    $t0,0($a1)        # B[i] = c + i
        lw    $t0,0($a0)        # A[i]
        add   $s2,$s2,$s0       # c += A[i]
        addi  $s0,$s0,1
        addi  $a0,$a0,4
        addi  $a1,$a1,4
        beq   $s0,$s1,Loop      # i != n
```

Krystian Bacławski