INF2270– Ukeoppgaver 8 - Fasit

Nøkkelord: Pipeline harzard, Cache miss og Page Failure

Oppgave 1: Cache miss og Page Failure

(a) Ta utgangspunkt i at du skal implementere et program som skal kjøre 1000 instruksjoner. Anta at 200 av instruksjonene trenger å aksessere minne. Klokkefrekvensen er satt til 3 GHz. Alle resterende instruksjoner som ikke trenger å aksessere minne trenger kun en klokkesyklus til å eksekverer. La oss anta at det ikke er noe som helst **penalty** for en **cache access** og at disse instruksjonene kan også gjennomføres i løpet av en klokkesyklus. Hvis programmet er velskrevet eller at programmereren er heldig, vil alle minne aksesseringer bli en **cache hit** (til og med den første minne aksesseringen også, da blokken er allerede er i cache før eksekvering). Beregn hvor lang tid det vil ta for at hele programmet er ferdig eksekvert.

En klokkefrekvens av 3 GHz gir en klokkesyklus på:

t = 1 / f
= 1 / 3GHz
= 1 / 3*10^9
= 0,33 * 10^-9
= 0,33 ns

Forutsatt ingen cache miss og alle instruksjoner eksekvereres innen 1 klokkesyklus så vil ett program med 1000 instruksjoner bruke:

0,33 ns x 1000 = 0,33 us

Ettersom vi får utført 1000 instruksjoner på 1000 sykler så er

CPI = 1000 / 1000
CPI = 1

(b) La oss gjøre oppgaven 1a litt mer realistisk. Anta nå at vi har en **cache hit** på 95% og at en **cache miss** vil trenge 20 klokkesykluser for å aksessere og lese inn minne. Hvor lang tid vil det nå ta hele programmet å bli ferdig? (I denne oppgaven skal dere ikke tenke på avanserte løsninger som tillater en pipeline med parallell eksekvering av instruksjoner samtidig som minne aksesseres.)

Med en hit rate på 95% av 200 instruksjoner av 1000, vil det føre til at 10 instruksjoner vil måtte ta penalty. Dette fører til at hver miss få en eksekveringstid på 20 klokkesykler. Dette betyr:

800 instruksjoner bruker 1 klokkesyklus
190 instruksjoner bruker 1 klokkesyklus
10 instruksjoner bruker 20 klokkesykler

Totalt går det med 800 + 190 + 200 = 1190 klokkesykler = 0,39 us.

CPI = 1190 / 1000 = 1,19

(c) La oss gjøre oppgaven 1b ytterligere mer realistisk. Anta nå at selve programmet er lagret i en **virtual memory page** som ikke er lastet inn i minne i det hele tatt, for eksempel at den må lastes inn fra hard-disken og inn i minne. Dette vil føre til en **penalty** for **page failure** på 1 million klokkesykler. Hvor lang tid vil det ta nå for programmet?

Dette vil føre til at vi får 1 000 000 klokkesykler i penalty og dermed blir totale tiden

1000000+1190 = 1.001.190 klokkesykler = 0,33 ms

(d) Ved å betrakte de foregående oppgavene hva vil du resonnere frem til vedrørende nyttigheten av å optimalisere instruksjonene for korte programmer? Hvordan vil resonnementet endre seg hvis dette korte programmet var en del av et større program og som kom til å bli kjørt/eksekvert flere ganger i løpet av det større programmet?

Slik om vist over er det lite å hente ved å optimalisere små programmer. Men hvis dette er en funksjon/routine eller kall som hele tiden kalles og kjøres flere ganger (f.eks i loop) som en del av et større program som er i minne lengre enn tiden det tar å laste den i minne så vil dette være verdt å optimalisere.

Oppgave 2: Pipeline Speedup

(a) **Speedup** for en pipelinet program er relasjonen mellom **execution time** for sekvensiell eksekvering over pipelinet eksekvering av programmet. Slik som forelest så vil det gå med litt tid til å initialt fylle pipelinet med instruksjoner, hvis vi ser bort fra hazarder nå så vil dette føre til at vi ikke kan klare å få en speed-up lik ***k***, men nesten. Beregn speed-up for følgende setting:

n = 100 k = 5 t = 1ns

Vurder så **penalty**en som ligger for å fylle pipelinet.

Ideelt sett så kan man forvente at speed-up tilnærmet k siden en ikke pipelinet program tar nkt sekunder til å eksekvere og en pipelined program kan nærme seg nt sekunder, derav vil optimalt speed-upen bli:

nkt / nt = k

Men problemet med at den initiale fyllingen av pipelinet vil føre til at:

kt + (n-1)t = (k + n – 1) t

Dette betyr at den nye formelen blir

nkt / (k+n-1)t = nk / (k+n-1)

For settingen gitt av oppgaven blir speed-upen bli:

= 100 x 5 / (5+100-1)
= 500 / 104
= 4,81

Hvorav CPI blir 104 / 100 = 1,04.

Merk at for høye n så er denne penaltyen ikke så dominerende.

(b) Nå skal vi se på effekten hazarder har for speed-up i en pipeline. La oss estimere **penalty** for eksempel for en **control hazard**. La oss anta at CPUen ikke har **branch prediction**. Anta at sannsynligheten for at ett gitt program inneholder **branch** instruksjon er Pb = 20%, mens sannsynligheten for at det blir utført en **branch** er Pt = 70%. La oss videre anta at det ikke er noe **penalty** for en **branch** som ikke blir utført, mens det for en **branch** som slår til vil det koste 3 klokkesykler i stedet for 1. For et program som består av 1000 instruksjoner, hva er speed-upen gitt av den beskrevne situasjonen? Beregn også CPI (**clock cycles needed pr instruction**).

CPI er gitt av:

(1-PbPt) x 1 + PbPt x 3
1 - PbPt + 3PbPt
1 + 2PbPt
1 + 2 (0,2*0,7)
1 + 2 (0,14)
1 + 0,28
1,28

nt * CPI = 0,42 us

(c) Hvordan vil beregningen i oppgave 2b endre seg hvis vi inkluderer en **branch perdiction** som kan forutsi 70% (Pc=70%) av tilfellene for en eksekvering av **branch**? Her kan dere anta at en korrekt forutsett **branch** koster ingen **penalty**, altså bruker bare 1 klokkesyklus. Ellers vil det koste 3 klokkesykler.

CPI blir forbedret til:

1 – Pb(1-Pc) + 3Pb(1-Pc)
1 – 0,2(1-0,7) + 3*0,2(1-0,7)
1 – 0,06 + 0,18
1,12

nt * CPI = 0,37 us