Superskalar mikroarkitektur

I jagten efter højere ydeevne kan man finde på at skrue op for ressourcerne. Hvis der er mere end en instruktion der udfører sin `X`-fase samtidigt, taler man om en superskalar maskine. Det er en naturlig følge af en simpel pipeline arkitektur, og de første superskalar arkitekturer kom da også kort efter disse. Valget af `X` fasen skyldes den indsigt at denne fase er særligt meget brugt og ofte af instruktioner som ikke har en dataafhængighed. Men for at have flere instruktioner er man også nødt til at udvide andre dele af arkitekturen.

En simpel 2-vejs superskalar kan derfor håndtere to instruktioner samtidigt i faserne `F`, `D`, `X` og `W`, men kun 1 instruktion samtidigt i fase `M`. Det er motiveret af at fase `M` er dyrere end de andre. Tilgengæld kan man knytte en delmængde af de mulige faser til forskellige klasser af instruktioner: f.eks. det er således at ikke alle har en fase `M`. Vi kan derfor gøres det ved at udvide M for de instruktioner vi ved bruger `M`. Man kan også undgå en fase `W` for instruktioner der ikke skriver til et resultat registerfilen.

Eksempel: Superskalar

Faser:

- Tilgængelige ressourcer: `F:2`, `D:2`, `X:2`, `M:1`, `W:2`
- Alle faser taget har en latenstid på 1
- `inorder(F,D,X,M,W)`

Begrænsninger på instruktioner:

	Instruktion	Faser	Dataafhængigheder
Aritmetik	`op a b`	`FDXW`	`depend(X,a), depend(X,b), produce(X,b)`
Læsning	`movq (a),b`	`FDXMMW`	`depend(X,a), produce(M,b)`
Skrivning	`movq b,(a)`	`FDXMM`	`depend(X,a), depend(M,b)`

Overvej afviklingen af følgende program:

```
                01234567      -- Vigtige dataafhængigheder
movq (r10),r11  FDXMMW        -- produce(M,r11)
addq $100,r11   FDDDDXW       -- depend(X,r11), produce(X,r11)
movq r11,(r10)   FDDDXMM      -- depend(M,r11)
```

```
subq $8,r10       FFFFDXW       --
subq $1,r12        FFFDXW       --
```

1. periode: Nu kan vi indhente og afkode både første og anden instruktion
2. periode: Begge instruktioner flyttes til afkodning og de to næste bliver indhentet. Her finder vi ud af at der er en dataafhængighed mellem de to første og sikre at anden instruktion blive stalled i `D`.
3. periode: Tredje instruktion (skrivning) flyttes til afkodning, men `subq` bliver stalled i `F`, da der ikke er plads i `D`.
4. periode: Sker ikke noget nyt.
5. periode: Venter på læsning
6. periode: Læsningen er færdig og alle instruktioner kan flyttes frem. Skrivningen behøver ikke at blive stalled, da afhængigheden til `r11` først er i fase `M`.
7. og 7. periode og frem forløber som forventet.

Hvis vi igen kigger på søjlerne i plottet, har ingen af disse flere forekomster af faserne end vi har ressourcer til rådighed. Der er altså højst 2 `F`'er, `D`'er osv., men kun højest et `M`. Dertil ser vi at søjlerne oppefra lister faserne bagfra og vi ved derfor at disse bliver udført in-order.

Hvis vi udregner CPI for denne maskine bliver den nu `CPI = 8/5 = 1,6`. Dette er signifikant mindre en vores simple pipeline maskine, hvilket var `CPI = 2,4`. I dette tilfælde kan vi se at den sidste subtraktion (den vi kunne flytte frem) endda fås gratis. Vi kan endda forvente at clock perioden for denne superskalar arkitektur er omkring det samme for vores simple pipeline; der er kun en mindre overhead for mere styring af de flere enheder.