Out-of-order eksekvering

Idéen om out-of-order eksekvering (også kaldet dynamic scheduling) af kode er så gammel som processoren selv. Det er dog ret svært at implementerer, så I mange år fandtes det kun som en mulig program optimering, som nævnt tidligere. Vi skal derfor helt frem til starten af 90'erne før det var en gængs del at mikroprocessorer.

Faser i program-rækkefølge - eller ej

Overvej afviklingen i følgende simple superskalar pipeline, hvor

	Instruktion	Faser	Dataafhængigheder
Aritmetik	`op a b`	`F--XW`	`depend(X,a), depend(X,b), produce(X,b)`
Læsning	`movq (a),b`	`F--XM-W`	`depend(X,a), produce(M+1,b)`
Skrivning	`movq b,(a)`	`F--XM`	`depend(X,a), depend(M,b)`

- Tilgængelige ressourcer: `F:2`, `X:2`, `M:1`, `W:2`
- Antal instruktioner under beregning: `F-X: 4`, `M-W: 1`

```
                01234567      -- Bemærkning
movq (r10),r11  F--XM-W       -- produce(M+1,r11)
addq $100,r11   F-----XW      -- depend(X,r11), stall F til r11 er klar
movq r9,(r14)    F--XM        -- ingen afhængighed, så hvorfor vente?  <----
BEMÆRK!
addq $1,r10      F--XW        -- ingen afhængighed
```

Bemærk at instruktion nummer 3 og 4 her får sin `X`-fase en clock periode tidligere end instruktionen før. På en måde overhaler instruktion nummer 3 og 4 altså instruktion nummer 2.

Det er ikke noget som bryder med antallet af tilgængelige ressourcer i vores superskalar maskine. Vi kan tælle faserne i søjlerne og alt er korrekt. Dog husker vi nu reglen for `inorder(F,D,X,M,W)`, som vi glemt at tage med. For at tjekke den skal vi sikre at faserne i hver søjle er ordnet modsat oppefra og ned. Dette er oplagt ikke tilfældet i clock periode 4 og 5 hvor vi jo ser det to instruktioner har overhalet.

Så for er sikre `inorder(F,D,X,M,W)` er vi nødt til at have:

```
                012345678     -- Bemærkning
movq (r10),r11  F--XM-W       -- produce(W,r11)
addq $100,r11   F-----XW      -- depend(X,r11), stall F til r11 er klar
movq r9,(r14)    F----XM      -- ingen afhængighed, men stall i F for inorder
addq $1,r10      F-----XW     -- ingen afhængighed, men stall i F
```

Vi kan for dette eksempel kun spare en enkelt clock periode, men vi kan ane at der findes meget mere ydeevne i form af mere parallelisme i udførelsen, hvis vi blot kan afvige fra inorder-kravet i en eller flere faser.

Det har man gjort for "special cases" i mange maskiner gennem årene, men de sidste 20 år er der etableret en mere generel "standard model" for out-of-order maskiner

Standardmodellen for out-of-order mikroarkitektur

Inorder og out-of-order

I denne model passerer instruktioner først i programrækkefølge gennem en indhentnings-pipeline til de ender i en skeduleringsenhed (scheduler). Derfra kan de udføres uden at overholde programrækkefølgen. Efter udførsel placeres resultaterne i en form for kø. Resultaterne udtages fra denne kø og fuldføres igen i programrækkefølge. Det gælder såvel for skrivninger til registre, som for skrivninger til lageret.

Vi kan beskrive det ved følgende faser der er fælles for alle instruktioner:

- `F` : Start på instruktionsindhentning
- `Q` : Ankomst til scheduler
- `C` : Fuldførelse

Og vi benytter lejligheden til at fjerne `W` trinnet fra beskrivelsen.

Lagerreferencer

I de hidtil beskrevne maskiner bruger både lagerreferencer og aritmetiske instruktioner fasen `X`. Det afspejler at man i simple maskiner foretager adresseberegning med den samme hardware som man bruger til aritmetiske instruktioner. I standardmodellen har man i stedet en dedikeret fase til adresseberegning, kaldet `A`. Denne skelnen mellem `A` og `X` gør at man kan begrænse `A` til at forekomme i instruktionsrækkefølge, mens de andre beregninger ikke har den begrænsning.

Instruktioner der skriver til lageret har et væsentlig mere kompliceret forløb i en out-of-order maskine sammenlignet med en inorder maskine. Disse instruktioner må ikke opdatere lageret før `C`, så i stedet placeres skrivningerne i en skrive-kø. Skrive-køen indeholder adresse og data som kan bruges til at udføre skrivningen senere, efter `C`. Instruktioner indføjes i skrivekøen

umiddelbart efter `A`. Da `A` er en fase der udføres i instruktionsrækkefølge, kan efterfølgende instruktioner der læser fra lageret sammenligne deres adresse med udestående skrivninger i skrive-køen, og hvis adressen matcher kan den tilsvarende værdi hentes fra skrive-køen. Instruktioner der skriver til lageret kan (skal) indsætte deres adresse i skrive-køen selvom den værdi der skal skrives endnu ikke er beregnet. Det tidspunkt hvor værdien kopieres til skrive-køen markeres `V`.

En lille out-of-order model

Her er en model af en lille out-of-order maskine:

	Instruktion	Faser	Dataafhængigheder
Aritmetik	`op  a b`	`F----QXC`	`depend(X,a), depend(X,b), produce(X,b)`
Læsning	`movq (a),b`	`F----QAM--C`	`depend(A,a), produce(M+2,b)`
Skrivning	`movq b,(a)`	`F----QAMVC`	`depend(A,a), depend(V,b), produce(V,a)`

- Tilgængelige ressourcer: `F:2`, `Q:2`, `X:2`, `A:1`, `M:1`, `V:1`, `C:2`
- Antal instruktioner under beregning: `F-Q: 8`, `M-W: 2`, `Q-C: 32`
- `inorder(F,Q,C,A)`
- `outoforder(X,M)`

Bemærk at udover at faserne `X` og `M` nu er erklæret out-of-order, så er der indsat en begrænsning på 32 instruktioner fra `Q` til `C`. Det vil sige vi tillader 32 instruktioner at være i forskellige faser mellem `Q` og `C`. Dette kaldes skeduleringsvinduet. Jo større det er, jo flere instruktioner kan maskinen "se fremad" i instruktionsstrømmen.

Bemærk også at til forskel fra alle de tidligere maskiner er der ikke længere noget krav om `X` skal følge i en bestemt afstand efter `Q`, eller at `C` skal følge i en bestemt afstand efter `X` eller `M`.

Disse begrænsninger ville give følgende udførelse

```
                012345678901234      -- Vigtigste bemærkninger
movq (r10),r11  F----QAM--C          -- produce(M+2,r11)
addq $100,r11   F----Q----XC         -- depend(X,r11), produce(X,r11)
movq r11,(r10)   F----QAM--VC        -- depend(A,r10), depend(M,r11)
addq $8,r10      F----QX----C        -- produce(X,r10)
movq (r10),r11    F----QAM---C       -- depend(A,r10), produce(M+2,r11)
addq $100,r11     F----Q----XC       -- depend(X,r11), produce(X,r11)
movq r11,(r10)     F----QAM--VC      -- depend(A,r10), depend(M,r11)
addq $8,r10        F----QX----C      -- depend(X,r10), produce(X,r10)
```

Med en gennemsnitlig ydeevne på 2 IPC.