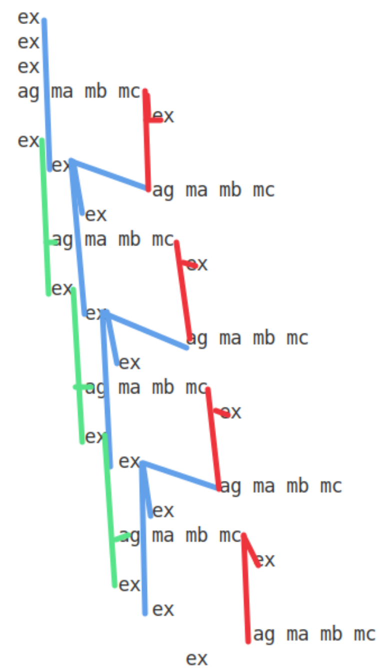
ILP - Instruction Level Parallelism

Hvis man betragter en sekvens af instruktioner vil man opdage at den oftest indeholder instruktioner, som ikke er afhængige af resultater fra de umiddelbart foregående. Disse instruktioner kunne i princippet udføres tidligere, samtidigt med instruktioner de ikke afhang af.

```
ADDI a5, a0, 0              R[15] <- ffffd4
ADD a0, a0, a1              R[10] <- ffffde
BEQ a1, zero, 100bc      {_}
LBU a4, 0(a2)               R[14] <- 62
BEQ a4, zero, 100b8      {_}
ADDI a2, a2, 1              R[12] <- 104ad
ADDI a5, a5, 1              R[15] <- ffffd5
SB a4, -1(a5)                        62        ->  Mem[ffffd4]
BNE a5, a0, 100a0        {T}
LBU a4, 0(a2)               R[14] <- 6c
BEQ a4, zero, 100b8      {_}
ADDI a2, a2, 1              R[12] <- 104ae
ADDI a5, a5, 1              R[15] <- ffffd6
SB a4, -1(a5)                        6c        ->  Mem[ffffd5]
BNE a5, a0, 100a0        {T}
LBU a4, 0(a2)               R[14] <- 61
BEQ a4, zero, 100b8      {_}
ADDI a2, a2, 1              R[12] <- 104af
ADDI a5, a5, 1              R[15] <- ffffd7
SB a4, -1(a5)                        61        ->  Mem[ffffd6]
BNE a5, a0, 100a0        {T}
LBU a4, 0(a2)               R[14] <- 62
BEQ a4, zero, 100b8      {_}
ADDI a2, a2, 1              R[12] <- 104b0
ADDI a5, a5, 1              R[15] <- ffffd8
SB a4, -1(a5)                        62        ->  Mem[ffffd7]
BNE a5, a0, 100a0        {T}
```



Fra RISCV til Dataflow

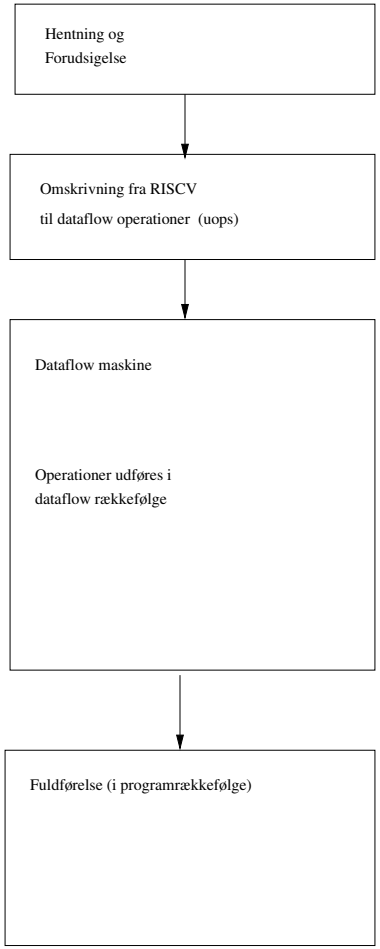
For at kunne bruge en dataflow maskine til at udføre programmer i RISCV maskinsprog (eller andet sekventielt maskinsprog) er det nødvendigt at oversætte RISCV instruktionerne.

Sådan en maskine er opbygget som 4 dele organiseret som en pipeline

- Hentning og forudsigelse af programforløbet. Det sker i program rækkefølge.
- Omskrivning af RISCV instruktioner til de operationer som dataflow maskinen forstår Ligeledes i program rækkefølge.
- Selve dataflow delen. Her udføres dataflow operationer i dataflow rækkefølge af flere små pipelines. De styres som beskrevet i tidligere forelæsning.
- Fuldførelse - her opsamles resultater produceret i dataflow rækkefølge og "præsenteres" i RISCV-program-rækkefølge

For at få god ydeevne skal man hurtigt kunne hente instruktioner og fylde dataflow-delen af maskinen med dem. Nu om dage gerne 4-8 instruktioner per clock-periode.

Fra RISCV til Dataflow

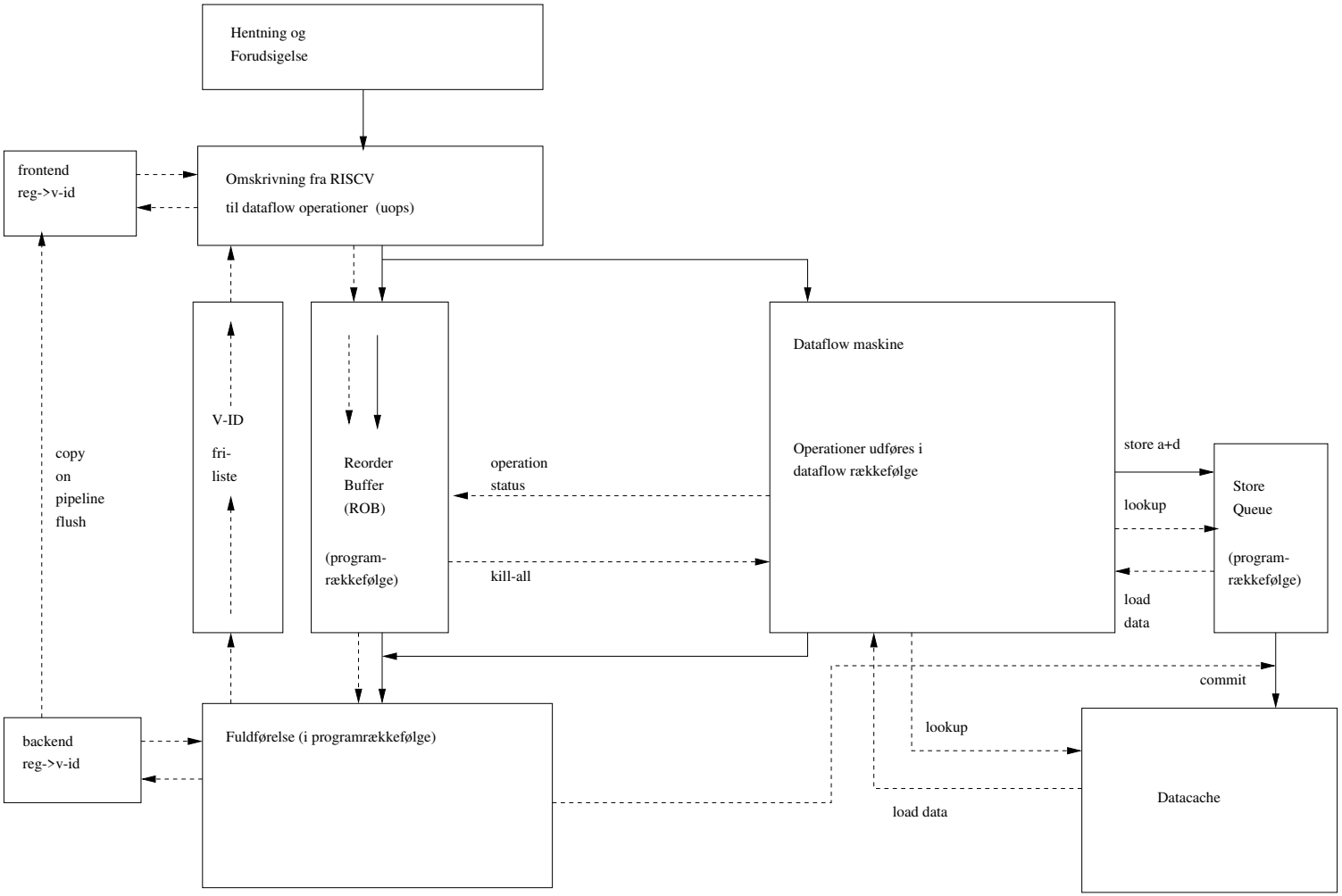


Dataflow vs sekventielt maskinsprog

En helt simpel dataflow maskine ved ikke noget om evt rækkefølge af instruktioner. For at kunne bruge den til at udføre et sekventielt program er vi nød til at tilføje nogle mekanismer der gør.

- I sidste forelæsning berørte vi kort behovet for spekulativ udførelse. Og dermed indirekte behovet for at kunne annullere instruktioner som er hentet og anbragt i dataflow maskinen, men som så viser sig ikke at skulle udføres.
- Vi tilføjede en STORE-kø, som holder spekulative skrivninger til lageret indtil de ikke er spekulative længere. Spekulative skrivninger må ikke nå til lageret.
- Det, at lagerreferencer har en rækkefølge (har sekventiel semantik) fører til at vi må sammenligne laggerreferencers adresse for at afgøre om der er en lagerbåren afhængighed (LOAD instruktioner skal se tidligere STORE instruktioner). Så vi gjorde STORE-køen søgbar for LOAD instruktionerne. Vi splittede også STORE i to operationer der kunne udføres uafhængigt af hinanden for at kunne beregne adresser så tidligt som muligt.

Fejlhåndtering (Exceptions)



Instruktions-flow

(Se diagram på tidligere slides samtidigt)

Vi opretholder program-rækkefølgen af instruktioner vha en ROB - "re-order buffer". Den kan opfattes som en lang liste af instrukioner. Nye instruktioner indsættes i den ene ende samtidig med at de sendes til dataflow maskinen. De ældste instruktioner udtages af den anden ende i programrækkefølge når de er udført af dataflow maskinen.

Derfor signalerer dataflow-delen til ROB'en når en operation er fuldført. Det sker selvfølgelig out-of-order. Instruktioner kan fejle (e.g. page-fault, division med nul) og hvis det sker, signaleres det også til ROB'en.

Den sidste del af maskinen, kaldet "Fuldførelse", tager instruktioner ud af ROB'en i programrækkefølge. Når en STORE instruktion (der ikke har fejlet) udtages, signaleres til STORE-køen og den ældste skrivning i køen udføres, dvs datacachen opdateres.

Hvis den ældste instruktion har fejlet tager fuldførelses-delen sig af fejlbehandling. (Beskrives senere)

RISCV Registre og V-ID'er

(Se diagram på tidligere slide samtidigt)

Værdier i dataflow maskinen har hver en unik ID (her kaldet V-ID).

Vi vedligeholder 2 tabeller der for hver RISCV register angiver den tilsvarende V-ID. Vi har en tabel i forenden af pipelinen og en i bagenden. Tabellen i forenden gælder når nye instruktioner omskrives og tilføjes til ROB'en. Tabellen i bagenden gælder når de ældste instruktioner skal fuldføres.

Under omskrivning fra RISCV til dataflow operationer oversættes hvert RISCV kilde-register reference til den tilsvarende V-ID ved opslag i tabellen. Til hvert destinations-register allokeres en ny V-ID fra fri-listen og tabellen opdateres med en ny binding fra RISCV-register til V-ID.

RISCV destinationsregister og V-ID tilføjes sammen med instruktionen til ROB'en således at man ved fuldførelse kan lave en tilsvarende opdatering af tabellen der. Ved fuldførelse bruges tabellen i fuldførelses-delen til at finde den tidligere V-ID for destinationsregisteret (som jo nu logisk set får ny værdi) og den tilføjes til V-ID frilisten.

Fejlhåndtering (Exceptions)

(Se diagram på tidligere slide samtidigt)

Hvis den instruktion der udtages har fejlet, så stoppes dataflow-maskinen. Alle instruktioner der og i forenden af maskinen annulleres. Instruktioner i ROB'en annulleres også, men deres V-ID'er nulstilles ikke. I stedet vil "Fuldførelses" delen af maskinen i de efterfølgende clock perioder gradvist tømme ROB'en for V-ID'er og tilføje dem til fri-listen.

Dernæst kopieres hele tabellen med register->V-ID bindinger fra fuldførelses-delen til den tilsvarende tabel i forenden af pipelinen. Dermed er alle resultater fra de annullerede instruktioner tabt, og for efterfølgende instruktioner fremstår det som om maskinen aldrig har arbejdet sig længere end til den fejlende instruktion.

Maskinen skifter til "exception processing" - dvs adresse på fejlende instruktion gemmes i et særligt register, der skiftes til supervisor/kernel/ priviligeret mode og instruktions-hentning omdirigeres til den relevante exception rutine.

Forudsigelser, korrigering

Vi kan bruge mekanismen til fejlhåndtering til at rydde op efter fejlagtige forudsigelser.

- Lad dataflow maskinen markere forudsigelser som korrekte/forkerte når den kan
- Når den ældste instruktion i ROB'en er markeret som en fejlagtig forudsigelse, kan vi nulstille dataflow delen og forenden af pipelinen.
- Tilbage er så kun at starte instruktions hentning fra den korrekte adresse.

Er det en god ide?

Forudsigelser, korrigering

Vi kan bruge mekanismen til fejlhåndtering til at rydde op efter fejlagtige forudsigelser.

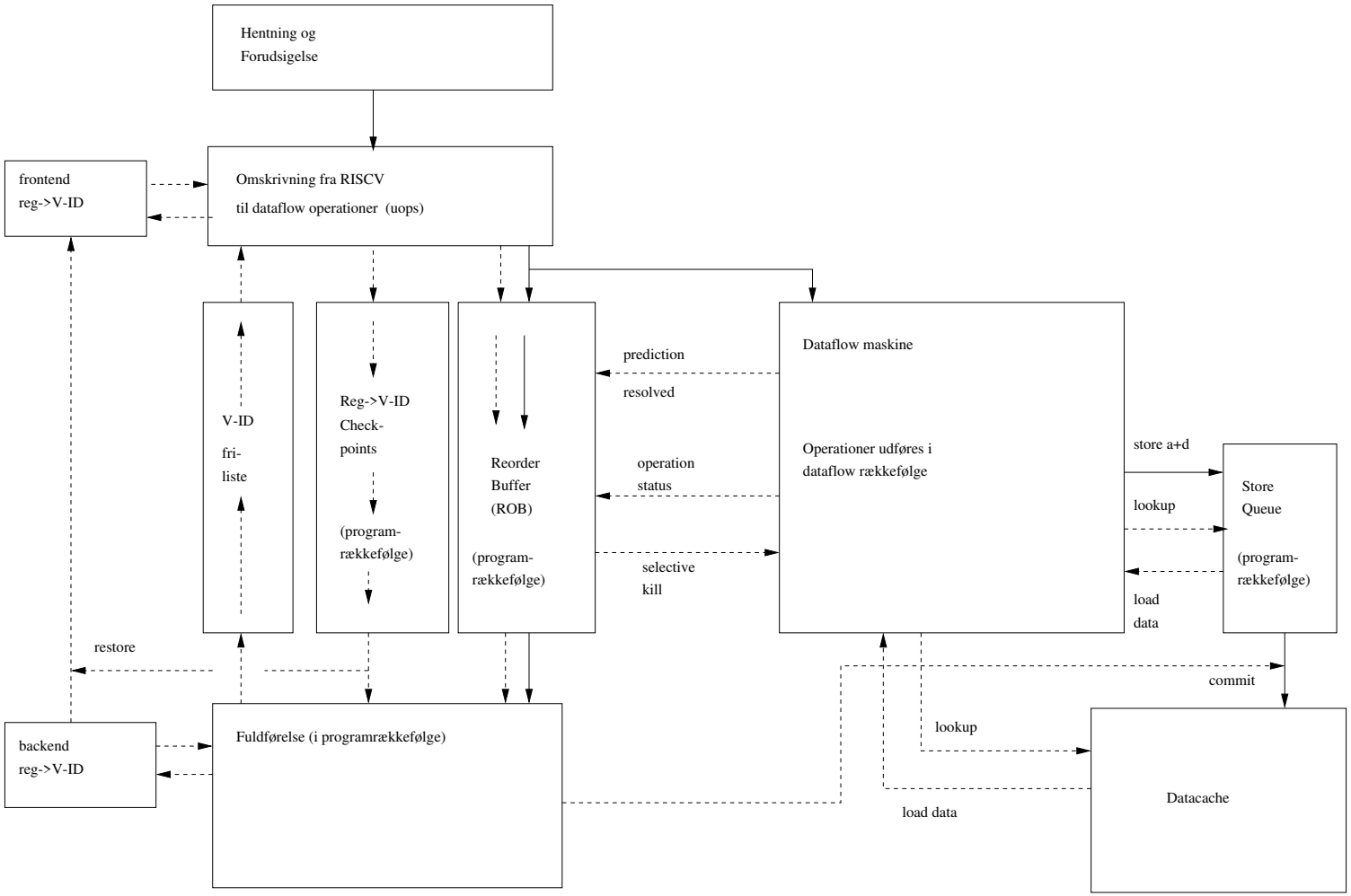
- Lad dataflow maskinen markere forudsigelser som korrekte/forkerte når den kan
- Når den ældste instruktion i ROB'en er markeret som en fejlagtig forudsigelse, kan vi nulstille dataflow delen og forenden af pipelinen.
- Tilbage er så kun at starte instruktions hentning fra den korrekte adresse.

Er det en god ide?

Desværre er den ikke god nok. Det vil for ofte tage for længe for en fejlagtig forudsigelse at blive den ældste og nå enden af ROB'en. Imens spildes værdifuld tid.

Vi vil hellere have en forbedret mekanisme så vi kan reagere hurtigere og korrigere en forkert forudsigelse så tidligt som muligt.

Selektiv annullering



Selektiv annullering (II)

Vi giver hver forudsigelse et fortløbende nummer. Og vi tilføjer til hver dataflow operation og hver instruktion i ROB'en nummeret på den sidst foretagne forudsigelse.

Når en fejlagtig forudsigelse rapporteres rundsendes den til alle instruktioner og operationer sammen med nummeret på forudsigelsen. Alle instruktioner og operationer sammenliger hver eneste clock periode det forudsigelsesnummer de har med sig med det rundsendte nummer. Herved kan det bestemmes hvilke der er ældre og hvilke der er yngre end den fejlagtige forudsigelse Alle der er ældre overlever, alle der er yngre annulleres.

Tilbage er at få korrigeret RISCV->V-ID tabellen i forenden af maskinen. Det skal gå hurtigt. En meget anvendt metode er at "checkpointe" tabellen hver gang der laves en forudsigelse. Tagne check points placeres i en særlig kø og ligesom instruktionerne har de nummeret på den sidst sete forudsigelse med sig. Når en forudsigelse fejler nulstilles alle de yngre checkpoints. Det yngste checkpoint der er tilbage bruges derpå til at retablere RISCV->V-ID tabellen i forenden.

Overgang

Hermed slut på gennemgangen af hvordan vi tilpasser vor dataflow maskine til et maskinsprog med sekventiel semantik.

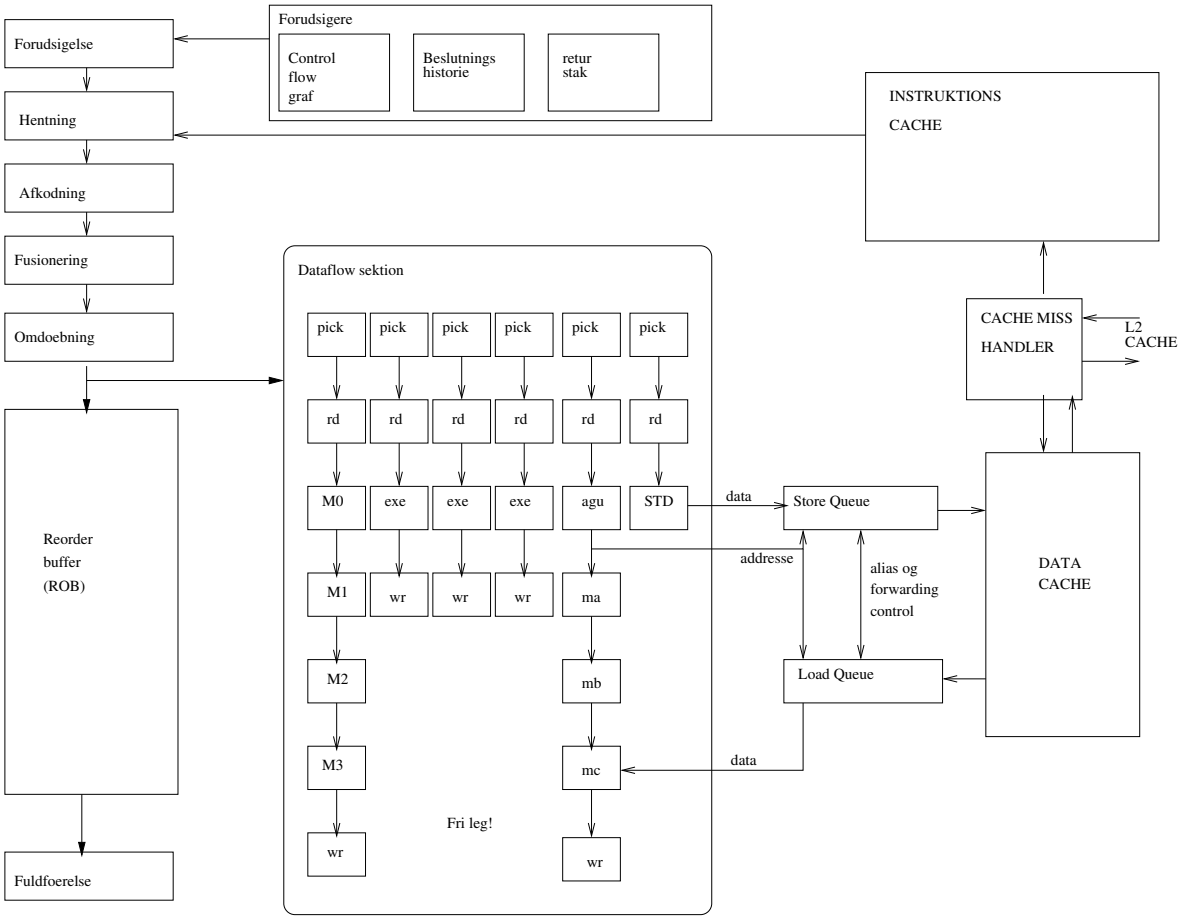
- Isolation / Skrive-kø
- Afbildning fra RISCV-registre til V-ID'er. Genbrug af V-ID'er
- Håndtering af fejlagtige forudsigelser

Det er et ret generelt framework. Samme overordnede struktur selvom vi ændrer på hvilke "små" pipelines vi vil have internt i vores dataflow maskine

Kommentarer?

Vi mangler nu bare at få fyldt maskinen med de rette instruktioner. Hurtigt.

En out-of-order mikroarkitektur



Forudsigelse af programforløb

Programforløbet forudsiges på basis af 3 lagerblokke allerforrest i pipelinen.

- BTB/CFG (branch target buffer) - lille cache der associerer PC med
  - Næste PC for et evt betinget hop
  - Næste PC hvis betinget hop ikke tages
  - PC-efter-kodeblok / Størrelse af kodeblok
  - Er der et betinget hop?
  - Er der kald?
  - Er der retur?
- RAS (return address stack)
- Forudsiger for betinget hop (beslutningshistorien)

Parallelt med at PC en sendes til instruktionscache for instruktionshentning, bruges den også til opslag i BTB og hop forudsiger. Hvis BTB'en siger at koden indeholder et betinget hop, så afgør hopforudsigeren om det skal tages eller ej. Hvis et hop ikke tages, så siger BTB'en om der er kald eller retur. Ved kald skubbes PC-efter-kodeblok på RAS, ved return tages næste PC fra RAS.

Når først det her kredsløb er "trænet" på basis af programmets opførsel, så kan det levere en ny PC for hvert "cycle"

Indkodning af CFG, Eksempel

```
Branch Target Buffer

entry   = 10074
next[0] = 100b0
next[1] = 10074
after   = 1009c
predict, call




entry   = 1009c
next[0] = xxx
next[1] = 10074
after   = 100ac
no predict, call

entry   = 100ac
next[0] = xxx
next[1] = xxx
after = xxx
no predict, ret
```

```
entry = 100b0
next[0] = xxx
next[1] = xxx
after = xxx
no predict, ret
```

```
Instruction Cache

 10074: addi    sp,sp,-16
 10078: sw      ra,12(sp)
 1007c: sw      s0,8(sp)
 10080: sw      s1,4(sp)
 10084: mv      s0,a0
 10088: li      a5,1
 1008c: bgeu    a5,a0,100b0 <fib+0x3c>
 10090: addi    a0,a0,-1
 10094: auipc   ra,0x0
 10098: jalr    -32(ra) # 10074 <fib>

 1009c: mv      s1,a0
 100a0: addi    a0,s0,-2
 100a4: auipc   ra,0x0
 100a8: jalr    -48(ra) # 10074 <fib>

 100ac: add     a0,s1,a0

 100b0: lw      ra,12(sp)
 100b4: lw      s0,8(sp)
 100b8: lw      s1,4(sp)
 100bc: addi    sp,sp,16
 100c0: ret
```

Dynamisk hop-forudsigelse (fra onsdag)

Dynamisk hop-forudsigelse opsamler data fra hoppenes historie og bruger det til at forudsige den fremtidige opførsel.

Den simpleste udgave betragter hvert hop for sig. Man knytter en to-bit tæller til hver hop, i praksis ved at lave en række af tællere og bruge nogle bits fra PC'en til at vælge en tæller. Hver tæller opsumerer hoppets historie:

```
00 hop ikke taget (strongly not taken)
01 hop almindeligvis ikke taget (weakly not taken)
10 hop almindeligvis taget (weakly taken)
11 hop taget (strongly taken
```

Hver gang et hop afgøres opdateres den matchende tæller, enten i retning mod "hop ikke taget", eller mod "hop taget".

Dette kaldes "local" hop forudsigelse - fordi man betragter hvert betinget hop adskilt fra de andre.

Korrelerende hop-forudsigelse

Hop er ofte korrelerede med andre hop. Det kan man udnytte ved at opsamle hoppenes historie. En simpel fremgangsmåde er at indkode historien i et skifte-register. Når et hop tages skifter man '1' ind i skifteregisteret. Når et hop ikke tages skifter man '0' ind.

Som før har man en tabel af to-bit tællere der opdateres på samme måde som beskrevet for lokale forudsigere.

For at lave en forudsigelse laver man et "hash" af skifteregisteret og PC'en og bruger det til at slå op i tabellen med tællere. Bitvis XOR er en fin hash funktion i det her tilfælde.

Denne forudsiger kaldes "gshare" og kan ofte levere mere end 90% korrekte forudsigelser.

Der findes andre og betydeligt mere omfattende forudsigere der fungerer endnu bedre. Generelt skal man ikke tro at man kan forudsige sine egne hop bedre end maskinen kan.

Se f.eks. https://team.inria.fr/alf/members/andre-seznec/branch-prediction-research/

Forudsigelse - Præcision vs Hastighed

Det kræver en hvis mængde lager til BTB og hopforudsiger for at få høj præcision. Desværre er store lager elementer langsommere end små, og høj præcision kan ikke fås i løbet af en enkelt clock cyklus.

Det er derfor normalt at man laver to forudsigelseskredsløb a la det tidligere beskrevne.

- Et der er optimeret til at kunne levere en forudsigelse pr clock. Det er så ikke så præcist.
- Et der er væsentligt større og optimeret til at være så præcist som budgettet tillader, men som er langsommere.

Det store kredsløb bruges så til at korrigere forudsigelserne fra det lille hurtige kredsløb.

Instruktions-hentning

Det vil vi ikke gå i detaljer med.

Det er mest et spørgsmål om at have en meget bred vej fra instruktions-cache til næste trin (4-8 instruktioner = 16-32 bytes)

Som tidligere nævnt i forbindelse med realistiske pipelines så antager vi at instruktionshentning gøres i tre pipeline trin, Fa, Fb og Fc uden at gå i detaljer med hvad der sker i de enkelte trin.

Omskrivning af instruktioner

Når en bunke RISCV instruktioner ankommer fra instruktionscachen bruges de til at generere tilsvarende dataflow-operationer. I tilfældet RISCV er det ret nemt, for mere komplicerede maskinsprog (x86) er det omfattende.

I vores eksempel-maskine her på CompSys fylder processen 5 pipeline trin. I virkelige maskiner kan det være såvel færre som flere. Vi bruger:

```
De  Decode, Afkodning.
    Tillige fastlægges afhængigheder mellem instruktionerne
Fu  Fusion.
    Hvisse instruktions-sekvenser slås sammen til en enkelt operation
Al  Allocate.
    Her allokeres resourcer og nye V-ID'er
Re  Rename. Omdøbning.
    Dataflow operationer med V-ID'er i stedet for RISCV registre produceres
Qu  Queue.
    De genererede operationer indsættes i dataflow-maskinen.
    RISCV instruktioner indsættes i ROB'en
```

Afkodning

Hver instruktion afkodes tilstrækkeligt til at vi senere kan bestemme hvilke resourcer den har brug for senere i maskinen.

Afhængigheder mellem instruktionerne indbyrdes bestemmes.

Fusionering

Fusionering omskriver instruktionspar til interne instruktioner som kan udføres hurtigere. Oftest skal begge instruktioner i et par have samme destinations register. Eksempler:

```
lui x3,immA
addi x3,x3,immB         -->  li x3,immA+immB

sll x5,x4,imm
add x5,x5,x6            -->  shiftadd x5,x4,x6,cnst

add x5,x4,x3            -->  lw2 x5,imm(x4,x3)
lw x5,imm(x5)
```

Det er selvfølgelig kun muligt, hvis maskinens datavej understøtter de nye interne instruktioner. I ovenstående indgår f.eks. en ALU der ved addition tillige kan skifte det ene input mod venstre.

Intel IA64 fusionerer ofte CMP og Bcc (compare og betinget hop). Det gør RISCV ikke, da det allerede er en enkelt instruktion.

Allokering

Efter fusionering ved man hvor mange dataflow operationer man producerer, hvor mange værdier operationerne vil producere og hvilke pipelines i dataflow maskinen der skal bruges.

- Der allokeres pladser i schedulerings-kredsløbene i de relevante pipelines til de nye dataflow operationer.
- Efter fusionering ved man hvor mange V-ID'er der er brug for til nye resultater. De allokeres fra fri-listen.
- Der allokeres plads i ROB'en til instruktionerne.
- Om nødvendigt allokeres plads i checkpoint-køen til et nyt checkpoint

Hvis (dele af) allokeringen ikke kan gennemføres må forenden af pipelinen stalle.

Omdøbning

Nu genereres de færdige dataflow operationer. Afhængigheder mellem dem er udtrykt som V-ID'er.

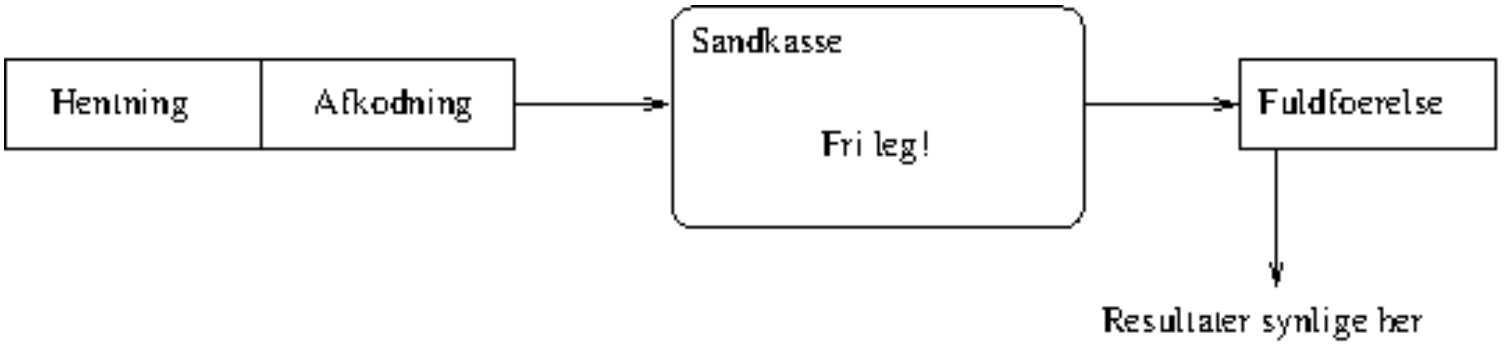
Indsættelse i køer

Slutteligt indsættes operationerne i schedulerings-køer i dataflow-maskinen og instruktioner indsættes i ROB'en.

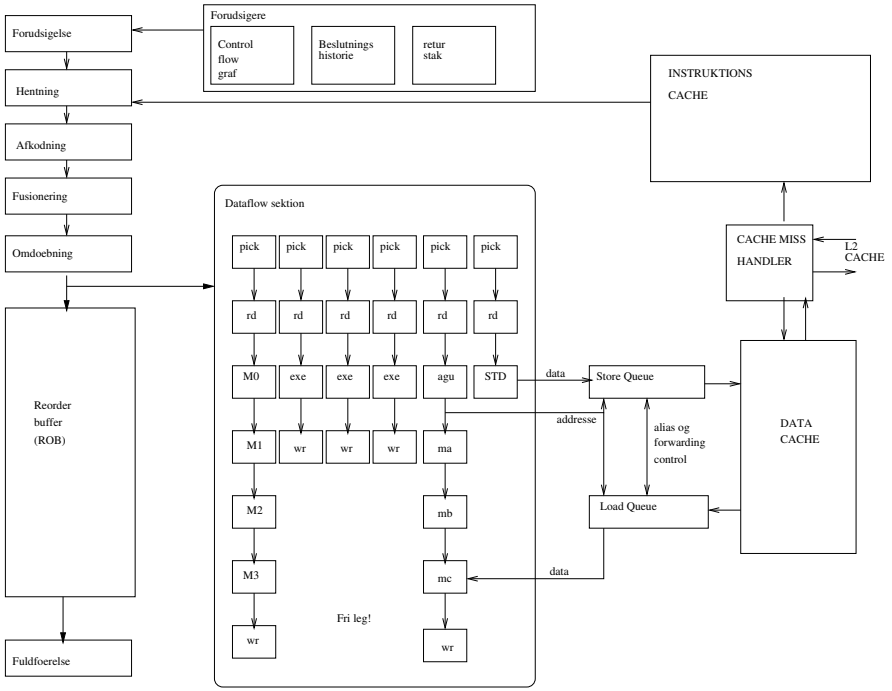
OOO - Opsamling

Out-of-order execution, eller "dynamisk udførelse" beror på tre principper:

1. Udførelsesrækkefølge fastlægges ud fra afhængigheder mellem instruktioner, ikke deres sekventielle rækkefølge i programmet
2. Spekulativ udførelse: Instruktioner udføres aggressivt i en "sandkasse", alle resultater/side-effekter skjules for omverdenen. "What happens in Vegas stays in Vegas"
3. Forudsigelse af programforløb gør det muligt at "fylde sandkassen" før vi kender programforløbet.



OOO - Mikroarkitektur - overview



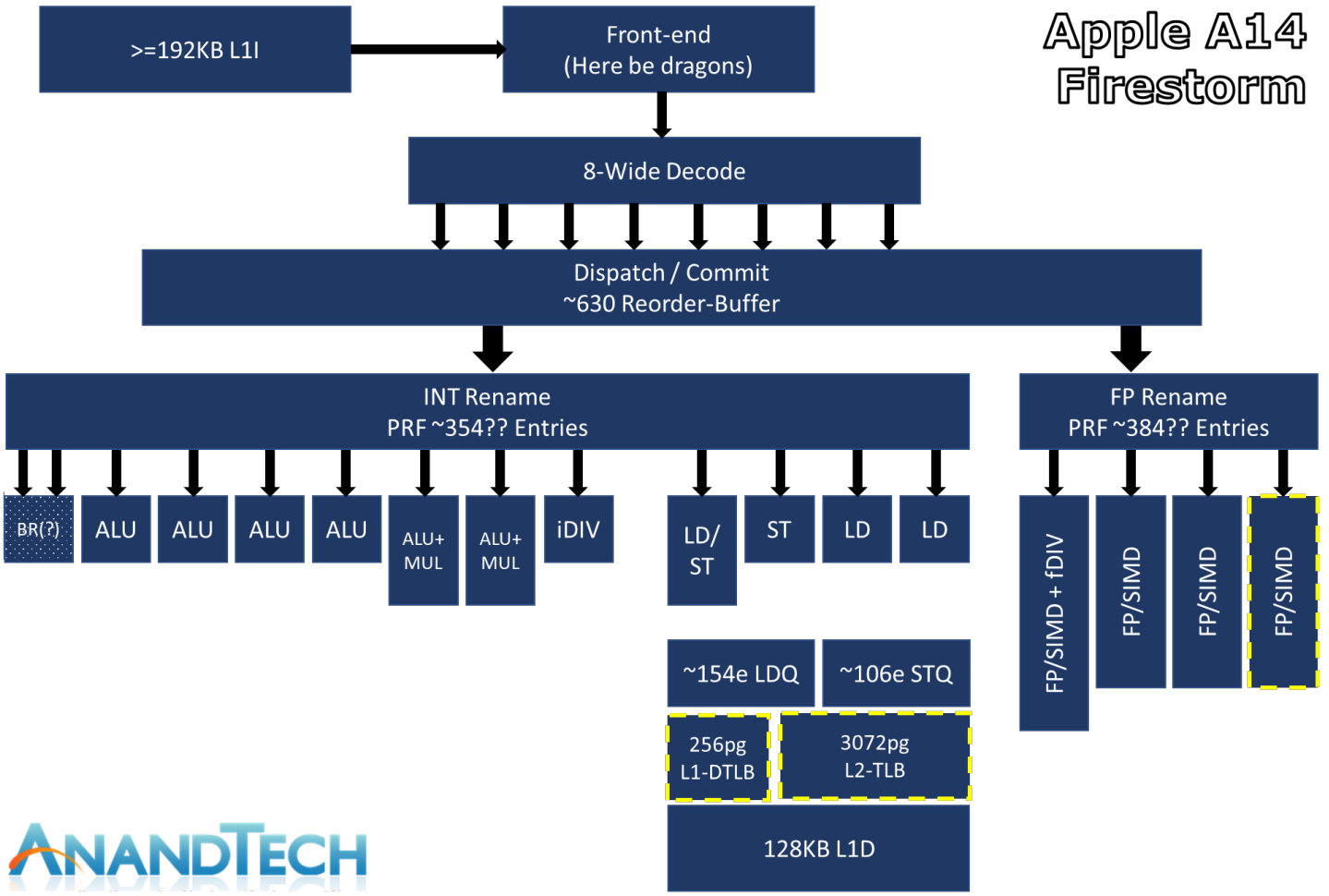

```
add  x12,x7,x3      Fa Fb Fc De Fu Al Rn Qu pk rd ex wb Ca Cb
lw   x11,8(x12)     Fa Fb Fc De Fu Al Rn Qu -- pk rd ag ma mb mc wb Ca Cb
addi x11,x11,400    Fa Fb Fc De Fu Al Rn Qu -- -- -- -- -- pk rd ex wb Ca Cb
addi x2,x12,32      Fa Fb Fc De Fu Al Rn Qu -- pk rd ex wb -- -- -- -- Ca Cb
```

Alle pipeline-trin forkortelser:

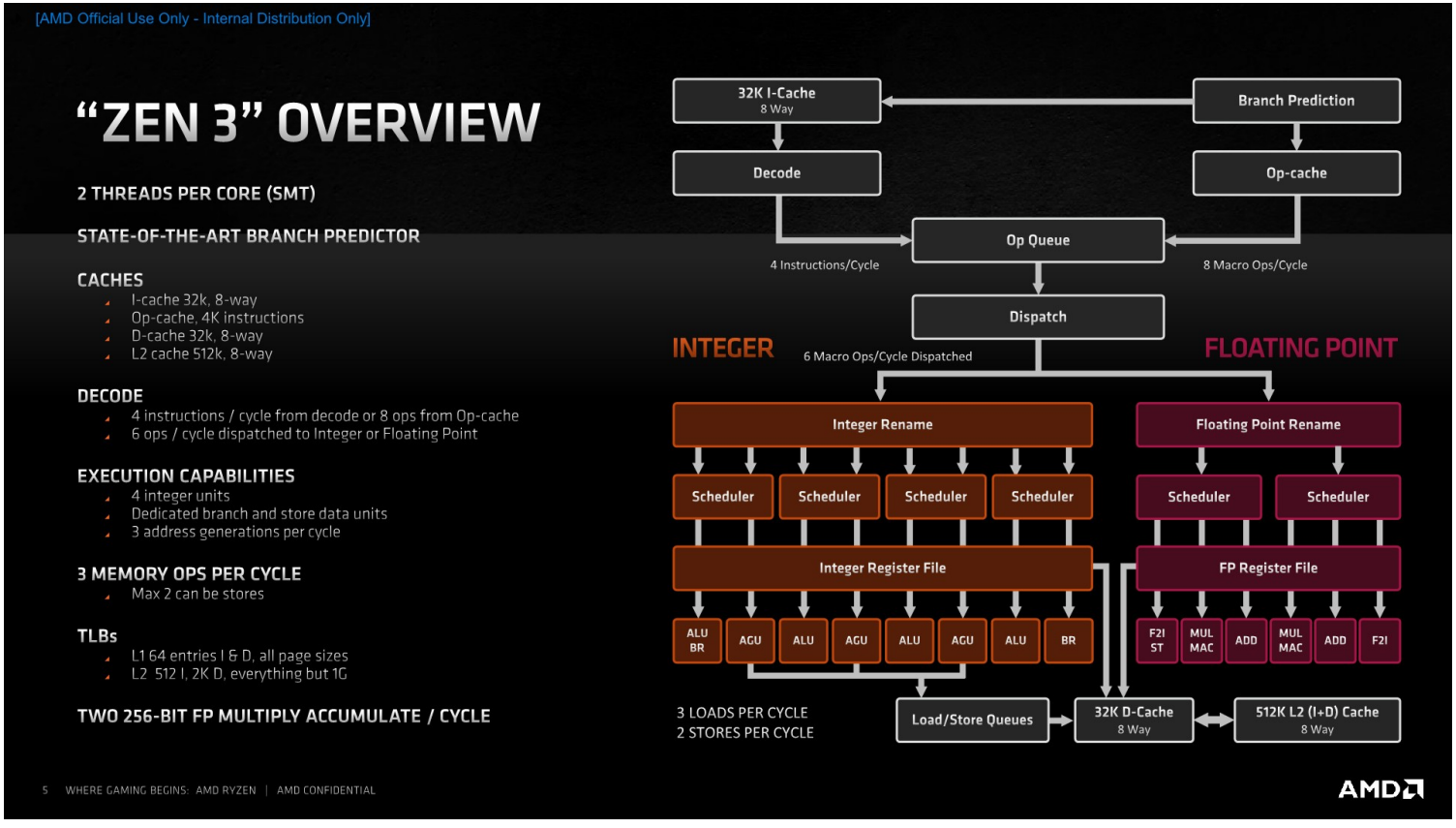
```
Inorder:                                    OutOfOrder/Dataflow
Fa  Fetch A                 pk  Pick (instruction)
Fb  Fetch B                 rd  Read (data)
Fc  Fetch C                 ex  Execute (arithmetic,control-flow)
De  Decode                  wb  Writeback (same as wr)
Fu  Fusion                  ag  Address Generate                    m0  Multiply 0
Al  Allocate                ma  Data cache A                        m1  Multiply 1
Re  Rename                  mb  Data cache B                        m2  Multiply 2
Qu  Queue                   mc  Data cache C                        m3  Multiply 3
Ca  Commit A
Cb  Commit B

add  x12,x7,x3      Fa Fb Fc De Fu Al Rn Qu pk rd ex wb Ca Cb
mul  x11,x12,x9     Fa Fb Fc De Fu Al Rn Qu -- pk rd ag ma mb mc wb Ca Cb
addi x11,x11,400    Fa Fb Fc De Fu Al Rn Qu -- -- -- -- -- pk rd ex wb Ca Cb
addi x2,x12,32      Fa Fb Fc De Fu Al Rn Qu -- pk rd ex wb -- -- -- -- Ca Cb
```

En rigtig ARM (Apple M1) - rekonstrueret



En rigtig x86 (AMD Zen 3) - overblik



Spørgsmål og Svar