Ein liten oversikt av kva som er viktigt å få med seg

Dette er ein oppsummering av det som er viktigast å få med seg. Kunnskapen er sjølvsagt eksamensrelevant, men mint like mykje kva som er nyttigt å kunne om datamaskiner for alle som kjem til å jobbe med datamaskiner eller i felt der datamaskiner er viktigt.

Huks at mykje av pensum må sjåst på som generelt. Eksempel på korleis maskiner virkar for å forklare ein generel mekanisme/virkemåte som skal gi kunnskap som gjer dykk i stand til å forstå alle maskiner. Datamaskiner er like (men ulike på detaljnivå).

Kapittel 1 og 2

Her er det mykje som blir omtala på eit høgt abstraksjonsnivå. Kva er ei datamaskin, korleis virkar dei og kva kan dei gjere. Viktige punkt:

- Kva er von Neumann arkitektur.
  - Arkitektur som kan utføre alle funksjonar ei Turing maskin kan berekne.
- Stored program computer
- Framdrift frå underligjande teknologi
  - Rele, transistor, IC -> Moores lov gir auka maskinvare resursar
  - Moores lov
- Paralelitet
  - ILP
  - Prosessornivå paralelitet (typar maskiner, e.g. SIMD, MIMD
- Minne hierarki
  - Kva og kvifor «Processor Memory Gap»
  - Statisk/Dynamisk RAM
  - Cach is King

Kapittel 3

Her er me på det lågast abstraksjonsnivået for faget. Må kunne forstå blockdigram og funksjonalitet til einingar som består av logiske portar

Viktige punkt oppsummert:

- Logiske portar (NAND, NOR, XOR, AND etc. For å kunne forstå skjemateikningar)
- Enkel logikk (alt kan lagast av NOR eller NAND)
- Buss og buss signal, korleis virkar ein buss
  - Legg ut ADR
  - Les/skriv
  - Adresserom (peikarar og atter peikarar)
  - Asynkron/synkron
  - Arbitrering (kva er det (to hovudtypar)
- Forstå blokk digram av f.eks:
  - CPU
    - Kva er ein instruksjon
    - Korleis utføres instruksjonar
  - Logikk i CPU einingar (Register, ALU etc)

- Adressedekoding og korleis/kva adresserom
- FSM ligningar, tabell, statediagram. Blir forelest etter microarkitektur.

Kapittel 4

Kapitel 4 har som mål å forklare mikroarkitektur. På dette nivået er prosessorar på blokskjemanivå. Målet er at dykk skal kunne bruke prosessormikroarkitektur til å forstå kva einingar ein prosessor er bygdopp av og korleis dei er kopla saman. Me har brukt IJVM som eksempel. Huks at dette er berre eit eksempel for å kunne sjå på detaljar:

- Oppbygjing av «datapath»
- Grensesnitt mot eksternt minne
- Styreeining

I IJVM er det gjort mange designvalg, f.eks. datapath der eine operanden er i H-register, microprogramert styreeining, for å få ein mikroarkitektur som passar til instruksjonssettet. Design valga for IJVM er også gjort utfrå resurshensyn, f.eks. aral og ytelse.

Begynnar med fyrste microarkitektur for IJVM (MIC1) Ved å forstå korleis styreeininga virkar (PC, MPC og utføringen av mikroprogram som gir ein sekvens av styreord (MIR)) styrer datapath for å få utført ein og ein mikroinstruksjon som tilsaman gir ein instruksjon. Ha innsikt i korleis IJVM utfører instruksjonar. Denne innsikta er noko eg håpar dykk kan bruke til å forstå mikroarkitektur generelt (for alle prosessorar).

Viktige punkt oppsummert IJVM (MIC1):

- Forstå korleis styreeining virkar
    - Program CounterPC (innheld instruksjon opCode (definert på ISA-nivå)
        - opCode verdi angir peikar til fyrste microinstruksjon i instruksjon.
    - MicroProgram Counter (innheld peikar til mikroinstruksjon i control store)
    - Micro Instruction Register MIR (styreord, alle signal for å styre datapath (Shift, ALU, B-bussog oppdatere MPC)
    - Control Store (Innheld microinstruksjonar (styreord, adressa til neste micro instruksjon og om det er ein branch instruksjon)
- Forstå korleis datapath fungerar
    - Bussar og kontroll av dei
    - ALU, shifter og kontroll
    - Register for minnetilgong
        - Dataminne: Memory Data Register (data som skal skrivast til minne (ved WR-signal) eller data som er lest frå minne (ved RD-signal))
        - Dataminne: Memory Adress Register (Peikar til aktiv minne adresse (for MDR))
        - Programminne: Program Counter PC (Peikar på instruksjon i programminne)
        - Programminne: Memory Buffer Register (opCode for instruksjon som er henta (fetch) frå programminne, og eventuelt operandar når naudsynt (huks dette er eit 8-bit register (opCode 8 bit, operand 8 bit (om gongen))

    - Dataflyt:
        - Valgt register legges ut på B-buss
        - Valgt ALU operasjon på operandar (B-buss eller H-register (ein operand). Eller B-buss og H-register (to operandar).
        - Skriv ALU resultat til eit eller fleire register via C-buss.

No har me gått gjennom eit eksempel på korleis ein prosessor kan være bygdopp, har kontroll på korleis IJVM virkar. Vidare er det metodar for å endre ytelse parameter (her auke hastigheit på instruksjonsutføring/berekning throughput). Huks at ytelse er gitt ut frå ytelsemål. Ytelsemål kan, som for for IJVM, være hastigheit men andre parameter er også mulig, f.eks. silisiumsareal eller effektforbruk.

Viktigt å få med seg: Når mikroarkutekturen endrast så endrast ikkje ISA. Alle variantar av IJVM kan utføre dei same instruksjonane.

IJVM mikroarkitektur endringar for å auke ytelse (kva og korleis):

- Innfører A-buss. Sparar klokkesyklar
    - Må ikkje lengre ha eine operand i H-register
    - Må utvide MIR med eit felt for å styre A-buss (som for B-buss)
    - A-buss utviding krever ekstra areal og logikk
- Innfører Instruction Fetch Unit IFU
    - Hentar neste instruksjon automatisk
    - Oppdaterar PC utan at me må bruke datapath (klokkesyklar) (auto increment av PC)
    - Har ein kø av instruksjonar og operandar (operandar frå programminne), slepp å vente
    - Men ved conditional branch må ein kunne oppdatere PC utan å bruke auto increment)
    - I IJVM IFU er det valgt å stoppe instruksjons henting til ein veit om branch betingelsar er oppfylt eller ikkje (sjå også pipelining)
    - IFU krever ekstra logikk og dermed ekstra areal og effektforbruk.
- Innfører pipelining
    - Delar opp innstruksjon i bitar som kan utførast uavhengig av
    - kvarandre. Kan då korte klokkeperioden til det lengste (treigaste) pipeline steget
    - .Huks frå tidligare: Fetch, Decode, Execute (legg og til writeback)
    - Brukar lachar/register til å skilje trinn i datapath
    - IJVM datapath pipeline; 3 trinn (A/B-buss, ALU, C-buss)
    - Utover datapath: Fetch, Decode ♣ Utvidar til 5 og 7 trinn
    - Pipeliner må handtere avengigheiter (RAW, WAW, WAR) sjå ISA
    - Pipeline må handtere branches («feil» instruksjonar i pipelina ved hopp)

Kapittel 5 og tillegg

ISA Opprineleg det einaste nivået, definerar instruksjonar logisk og minnemodellar.

Instruksjonar:

- Type (instruction format (fig. 5.9 bok), antal operandar, 0 adr, 1, adr, 2, adr, 3, adr osv

- Adresseringsmodi
- Korleis operandar handterast, f.eks:
    - Imidiate
    - Direct
    - Register
    - Register Indirect
    - Indeksert

Instruksjonslengde. Variabel eller Fast.

- Lengde og oppbygging av instruksjonar
- Koding av instruksjonar, f.eks. (tabell forelesing 17, fig 5.12 bok)

Instruksjons typar

- Data flytt
- Manipulering (Dydadic, monadic)
- Samanligning, betinga hopp
- I/O (inkludert interrupt)

6.1 og tilegg, div

Virtuelt minne. Page, representasjon av virtuell/fysisk adr.

Flyttal: representasjon av nummer i datamaskiner. Korleis det kan gjerast.

Avhengigheiter.