Alpha arkitektur --- innledning ---

- **Kilder:**
  - ***"Alpha AXP Architecture"***, Richard L. Sites, CACM februar 1993, side 33-44
    - » den viktigste av de to artiklene
    - » detaljer om instruksjonssett og formater ikke pensum
  - ***"The Alpha Demonstration Unit: A High-Performance Multiprocessor"***, Charles P. Thacker et.al., CACM februar 1993, side 55-67
    - » mindre viktig
    - » mye lavnivå detaljer som ikke er relevant i faget
    - » pensum: "cache coherence" + det som foreleses
- **Forelesnings-innhold**
  - Forskjell mellom arkitektur og implementasjon
  - Mål for arkitekturen
  - Oversikt over arkitekturen
  - Forskjeller mellom Alpha og andre RISC'er
  - Noen sentrale tekniske løsninger
  - Litt om den første implementasjonen
  - Litt om Alpha Demonstration Unit (ADU)

Forskjellen mellom arkitektur og implementasjon

- ***"Arkitektur er en beskrivelse av oppførselen til alle mulige implementasjoner"*** [Sites93] side 33
- **Arkitektur**
  - egenskapene og oppførselen til en datamaskin slik en maskin-kode programmerer ser det
  - lang levetid ønskelig
    - » gode arkitekturer lever lenge
    - » beskriv "hva men ikke hvordan"
    - » en for detaljert beskrivelse kan være uheldig
      - jfr. Leslie Valiant's argumentasjon om krav til en "bridging model"
- **Implementasjon**
  - den aktuelle maskinvare-løsning som gir en mulig realisering av arkitekturen
  - kort levetid som god implementasjon
    - » ny teknologi (og annet) gir stadig muligheter for nye og bedre implementasjoner

Mål for arkitekturen

- **Ledende på ytelse ("DEC catchup")**
  - Superskalar og dyp pipeline (superpipelined)
  - 64 bits adresse-område
  - Støtte for multiprosessorsystemer
    - » store og små
- **Generell**
  - Ulike OS
    - » OSF/1 (Unix) og VMS
  - Ulike språk
  - Lett overgang fra VAX og MIPS arkitekturen
    - » Binær-translasjon
      - valgt framfor "HW-compatibility-modes" for å maksimere kjøretidshastighet
- **Lang levetid**

Lang levetid (Eng. longevity)

- **Ønsker en arkitektur som varer 15-25 år**
  - unngå løsninger som kan bli en begrensning i løpet av denne tidsperioden
    - » f.eks. valg av full 64 bits arkitektur
- **Datamaskiner er blitt 1000 x raskere de siste 25 år**
- **Ønsker 1000 x ytelsesforbedring de neste 25 år, hvordan?**
  - 10 x klokkefrekvens = *f* (halvlederteknologi)
  - 10 x
    - » superskalar ("multiple instruction issue")
    - » pipelining
    - » bedre kompilatorer
  - 10 x multiprosessering
    - » i forhold til en en-prosessor-maskin
- **Skalerbar arkitektur**
  - tillate ulike implementasjoner (arkitektur)
  - fleksibilitet
    - » "PALcode"
    - » romslige "felt-størrelser"
      - 64 bits virtuell adresse
      - ledige instruksjonskoder for utvidelser

Alpha arkitektur: Oversikt (1/2)

- **64 bit "Load/Store"-arkitektur**
  - dvs. typisk RISC
    - » data flyttes mellom lager og registre uten modifisering (beregninger)
    - » alle beregninger utføres på registre
- **32 bit instruksjonslengde (fast)**
- **Datatyper:**
  - 8, 16, 32, 64 bit heltall
  - VAX 32 og 64 bit flyttall
  - IEEE 32 og 64 bit flyttall
  - standardisert representasjon av 32 bits verdier i 64 bits register
    - » øverste bit som "fat bit"
    - » ..s 37/38
- **To-delt register fil**
  - 32 x 64 bit registre
  - 32 x 64 bit flyttallsregistre
  - tallet 32 er fast bestemt av arkitektur
  - Kompilatorer kan gi en vesentlig ytelsesforbedrig ved bruk av 32 framfor 16 registre, men
    - » tilsvarende forbedring er ikke vist ved overgang til 64 registre

Alpha arkitektur: Oversikt (2/2)

- **64 bit programteller**
- **32 bit syklusteller**
  - for ytelsesmålinger
- **FPCSR**
  - Floating Point Control and Status Register
- **64 bit virtuelle adresser**
  - benyttes i hele arkitekturen
  - mindre implementasjoner tillates
- **Enkelt instruksjonsformat**

Multiprosessor-støtte, synkronisering

- **ldx_l (load locked) og stx_c (store cond.)**
- **Ide hentet fra MIPS R4000**
- **Prinsipp / algoritme**
  - Tilsvarer "optimistisk synkronisering" [Hwa93]

  1) hente - låst (ldx_l)

  2) modifiser (i registre)

  3) betinget lagring (stx_c)
  - **atomisk oppdatering:** Hvis hele sekvensen 1)-3) skjer uten avbrudd eller forstyrrelse av andre prosessorer vil stx_c skrive ut den modifiserte verdien, samt legge igjen informasjon om dette i registre som programmet kan teste etterpå.
  - **mislykket**: Hvis sekvensen avbrytes vil stx_c ikke skrive ut og programmet må gjøre nytt forsøk
- **Motivasjon**
  - vurdert som å være den eneste metoden som vil passe de prosessor-hastigheter en vil se de neste 10-20 år
- **Fordeler**
  - hele sekvensen i hurtigbuffer på prosessor-brikken
  - mange uavhengige slike sekvenser kan utføres samtidig

Multiprosessor-støtte, delt lager

- **Lager barriere instruksjon**
  - lese/skrive ordning bør ikke være strengere enn nødvendig
  - la kompilator/programmerer avgjøre når en trenger ordning av operasjoner, og gi frihet ellers
- **Prosessor nummer (ID)**
  - nødv. for at et program skal vite hvilken prosess & prosessor det kjører på

Arkitektur: Mulige flaskehalser (1/2)

- **Felles ressurser (f.eks. registre) i prosessoren**
  - Unngå flest mulig av disse (både synlige og skjulte)
  - Eksempler
    - » statusflagg-register
    - » register for multiplikator-kvotient
  - I en superskalar prosessor med pipelining har vi mange instruksjoner "underveis" samtidig
    - » "side-effekter" på felles-registre
    - » instruksjoner som skal startes må "sjekkes mot" pågående bruk av felles ressurser
      - dette forsinker
      - bør ha færrest mulige
      - vanlige registre kan ikke unngås
  - Bruk generelle registre til den samme informasjonen om nødvendig
    - » kan da velge blant ledige registre framfor å vente på en felles ressurs
  - Generelt: gjør instruksjonen mest mulig uavhengig av hverandre (reduser felles ressurser og sideeffekter)
    - » Dette er en instans av det generelle prinsippet: "lokalitet gir parallellitet"

Arkitektur: Mulige flaskehalser (2/2)

- **Byte operasjoner mot lager**
  - lageret er konstruert for å aksesseres ord for ord
  - byte-operasjoner mot lageret krever
    - » logikk mellom register-blokk (Eng. register file) og lagergrensesnitt
    - » evt. støtte i lagersystemet
    - » begge deler kompliserer og gjør alle lageraksesser langsommere
    - » prosess: les ord, sjekk "paritet", modifiser, generer "paritet", skriv ord
    - » mange instruksjoner kan ønske å gjøre dette samtidig
  - alternativ hvor prosessor-lager-grensesnittet i mindre grad blir flaskehals:
    - » les hele ord inn i registre, modifiser (i parallell) på registre, skriv ut fra registre
    - » forventes (på sikt) å gi raskere byte-manipulerings rutiner enn prosessorer som har en spesial-instruksjon for byte-operasjoner mot lager

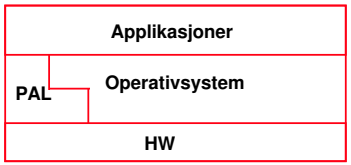
"No first implementation artifacts"

- **"Clean features" vs. "first implementation artifacts"**
  - "istf. lag en prosessor-brikke, selg den, lag en ny (med utvidelser, forbedringer og nedover-kompatibilitet) heller (Alpha) lage en arkitektur som løser alle hoved-problemer en kjenner til, selv om det gjør realisering av den første prosessor-brikken vanskeligere"
  - ((litt sant, litt markedsføring?))
- **Eks.: PALcode og binær-translasjon**
  - *"... gave us the luxury of designing the Alpha AXP architecture from a "clean sheet of paper""*

Hva er nytt/forskjellig?

- **Barriere instruksjon istedetfor streng ordning av lese/skrive operasjoner**
- **Hopp-prediksjon, "prefetch" og betinget flytt istedetfor forsinket hopp (Eng. delayed branch)**
- **"Multiple-issue" (bredde-parallellitet) i stedet for spesialoperasjoner som f.eks. MULADD**
- **"Hazardfree" arkitektur**
  - kompilator har ikke ansvar for korrekt utførelse
  - alle prosessor-implementasjoner er slik at "effekter av foranliggende instruksjoner er klare for (benyttes av) etterfølgende instruksjoner"
- **Minimal global tilstand**
  - viktig ved kontekstbytte
- **Ingen modus-bit**
  - skifte av modus-bit krever ofte tømming av instruksjons-pipeline og ny oppstart av denne
  - minimaliserer maskinvare-kompleksitet og øker tiden hvor pipeline er fullt utnyttet
  - typisk modus-info kodes i instruksjonene
    - » F.eks. IEEE_FAD og VAX_FADD

Fleksibilitet v.h.a. "PALcode"
PAL = "Priviliged Architecture Library"



- **Ide fra PRISM**
  - et tidligere RISC prosjekt hos DEC
- **PALcode =**
  - spesielle "priveligerte" programvare-rutiner
  - spesialinstruksjon for å kalle disse
  - kjører med avbrudd avskrudd
  - utfører feilhåndtering, lagerhåndtering, kontekts-bytte m.m.
  - aksess til prosessor-registre
- **Ulik PAL-kode for ulike OS gir**
  - OS ser en "mer passende" maskin
  - HW kan konstrueres "mere på fritt grunnlag"

Hensyn til parallellitet, pipelining og lageraksesstid

- **Upresise unntak/avbrudd (Eng. exceptions)**
  - Ide hentet fra Cray-1
  - **presist avbrudd**: ingen instruksjoner som etterfølger avbruddet blir utført før avbruddet er ferdig behandlet
    - » lett å realisere på en langsom, gammeldags prosessor
    - » vanskelig på en moderne prosessor som har flere instruksjoner under utførelse samtidig
      - krever "undo"-logikk m.m.
  - gir enklere/raskere pipeline
  - presise avbrudd kan oppnås ved å spesifisere det eksplisitt bare der det er nødvendig
    - » TRAPB - instruksjon for å "vente på" unntak
- **Spesialinstruksjoner**
  - TRAPB
  - MB
    - » for å "vente på" fullføring av lageraksess
  - FETCH
    - » "hint" til lagersystemet om at det kan være "lurt" å hente en gitt hurtigbuffer-blokk
      - gjør det lettere å lage "intelligent" kompilatorer

Utvalgte problemer: --- Problem 1: Hopp ---

- **Hopp, hvordan få minst mulig dødtid ved hopp?**
- **Valgt løsning:**
  - Hopp-prediksjon, hint og regler for betingete hopp
    - » Bakover hopp? ==> antar hopp
    - » Forover hopp? ==> antar ikke hopp
  - Hopp "er merket som" kall, retur, el. ingen-av-de-to (mhp. subrutiner)
    - » kan "holde inne" en stakk av sannsynlige retur-adresser
  - Adresse hint ved "computed goto"
    - » 14 ubrukte bit i instruksjons-ordet gir et antall MSB av den mest sannsynlige hopp-adressen
      - "prefetch-hint"
  - Betingete flytt-instruksjoner for å fjerne "små" hopp
    - » f.eks. teste R1, og ut fra resultat kopiere R2 til R3 el. ikke
    - » f.eks. beregne MAX el. MIN av to verdier
- **Forkastet løsning**
  - Forsinkede hopp ("delayed branch")
    - » en god løsning i 1988
    - » umulig å velge en rett størrelse på "delay-slot" for alle implementasjoner (forskjellige pipelines og forskjellig grad av superskalaritet)

Problem 2: hvordan få til bredde-parallellitet ("multiple issue") ?

- **Valgte løsninger**
  - reduser avhengigheter mellom instruksjoner
  - ingen "hazarder"
  - ingen spesialregistre
  - ingen presise aritmetiske unntak
  - minimalisere global tilstand
    - » IEEE-kontroll i instruksjonene
- **Forkastede løsninger**
  - Fast antall instruksjoner/bredde for instruksjons-oppstarting (VLIW)
  - Forsinket hopp (Eng. delayed branch)
  - Modus bit

Problem 3: Økende lageraksesstid i forhold til CPU syklustid

- **Valgte løsninger**
  - FETCH instruksjoner som henter blokk til hurtigbuffer før den trengs
  - Ingen gitt ordning av lese/skriveoperasjoner i multiprosessor systemer (tillater treff samtidig som lager-systemet betjener en bom)
  - Har istedet barriereinstruksjon, “memory barrier” (MB)
    - » inspirert av IBM System/370
- **Forkastede løsninger**
  - Gitt ordning av lese/skrive-operasjoner
  - Synlige hurtigbuffer-effekter (for bruker)

Diverse

- **Todelt registerfil (heltall og flyttall)**
  - Alternativ: Kombinert registerfil
    - » mer fleksibelt m.h.p. register-utnyttelse i program med skjev register-type fordeling
    - » mer fleksibelt for oversending av en blanding av heltalls og flyttalls subrutine-parametre i registre
  - **Todelt registerfil:**
    - » lettere å lage en implementasjon for bare heltall
    - » enklere grensesnitt mot hver registerfil gir hurtigere aksess
- **Kanonisk format for 32-bits data**
  - Standardisert format for representasjon av 32 bits data i 64 bits registre
    - » kan utnyttes i prosessor-implementasjonen ved å bruke de samme funksjonelle enheter på en effektiv måte på ulike ordlengder

EV-4, første implementasjon av Alpha

- **EV-4 (internt kode-navn)**
  - Extended VAX, 4-generasjon CMOS-prosess
  - DECchip 21064
- **Data**
  - 200 MHz (5 ns)
  - CMOS (0.75 mikro-m)
  - 14 x 17 mm
  - 1.68 mill transistorer
  - 431 pinners PGA
    - » 291 signaler, resten strømforsyning
  - 30 Watt ved 3.3 Volts spenning
  - superskalar av grad 2
  - 8 kB instruksjons-hurtigbuffer (primær)
    - » direkte avbildning
  - 8 kB data-hurtigbuffer (primær)
    - » direkte avbildning
  - 7 trinns heltalls-pipeline
  - 7 trinns lese/skrive -pipeline
  - 10 trinns flyttalls pipeline
  - støtte for ekstern cache

EV-4 CPU-blokk-diagram

Alpha Demonstration Unit (ADU)

- **“Proof of concept”**
- **Arbeidsbenk for avlusing av maskinvare og programvare**
- **Minimaliser kompleksitet og makimaliser ytelse**
  - kostnad uinteressant
- **Består av**
  - bakplan-buss
  - prosessor-moduler med hurtigbuffer
  - lagermoduler
  - I/O-moduler

ADU

- **Bakplanbuss**
  - konstruksjon er meget sentral for ytelse i multiprosessor
  - enhver operasjon skjer en fast tidsforsinkelse etter forespørsel
    - » forenkler
  - to uavhengige transaksjoner kan betjenes samtidig
    - » bedret ytelse
  - 64 adresse/data linjer
  - 14 bit for feilkorrigerende kode
  - 2 signaler for hurtigbuffer-koherens
  - flere detaljer gitt i artikkelen
- **Prosessor-modul**
  - Alpha CPU
  - 256 kB sekundær-hurtigbuffer
  - masse test-logikk
    - » store deler av systemet ble testet før Alpha-chip’en var klar
  - flere detaljer gitt i artikkelen

ADU: hurtigbuffer-koherens

- **To hovedprinsipper for hurtigbuffer-koherens; skriv-oppdater & skriv-ugyldiggjør**
  - Skriv-oppdater er best ved
    - » utstrakt deling av data mellom prosessorer
  - Skriv-ugyldiggjør er best ved
    - » prosess-migrering (flytting)
- **Spesifikasjon av protokoll**
  - ca. 15 sider uformell prosa
  - virkelig spesifikasjon: parallelt program (Eng. multithreaded)
    - » beskriver alle grensesnitt på signal-nivå
    - » kan simulere forventet oppførsel
      - redusert “turnaround” tid under konstruksjon
      - ytelsesmålinger før maskinvaren var klar
      - redusert utviklingstid

ADU koherens-protokoll

- **Oppdatering & ugyldiggjøring (hybrid)**
- **To-nivå hurtigbuffer**
  - “on-chip” (primær), større sekundær-hurtigbuffer
- **Sekundær-buffer med kombinasjon av “Write-Through (WT)” og “Write Back (WB)”**
  - initielt skriv-oppdater; er blokken delt med andre utføres WT for å oppdatere andre kopier. Er blokken ikke delt med andre er WT unødvendig; bruker WB.
  - mottar oppdatering fra annen prosessor:
    - » a) ugyldiggjøre egen kopi og rapportere at den har gjort det. Hvis deling opphører kan oppdaterende blokk slutte å bruke WT
    - » b) oppdatere egen kopi, og rapportere det. Oppdatering vil fortsette
    - » Alternativ a) velges hvis blokken *ikke* er i primær-buffer. (Hint som antyder lav grad av deling).
    - » Alternativ b) velges hvis blokken er inne i primær-buffer
      - primær-buffer-kopi ugyldiggjøres og sekundær-buffer-kopi oppdateres
      - hvis blokken deles *aktivt* vil primær-buffer kopi bli lest inn på nytt før neste eksterne oppdatering, slik at deling fortsetter !

ADU koherens protokoll implementering

- **Samarbeid mellom primær- og sekunder-buffer**
  - valg av strategi i sekundær-buffer avgjøres ved oppslag i en tabell som viser hvilke blokker som er lagret i primær-buffer
    - » uten å forstyrre CPU
- **Tilstands-diagram og forklaring i artikkelen**

Copyright © 1995 Lasse Natvig, IDT/NTH/UNIT