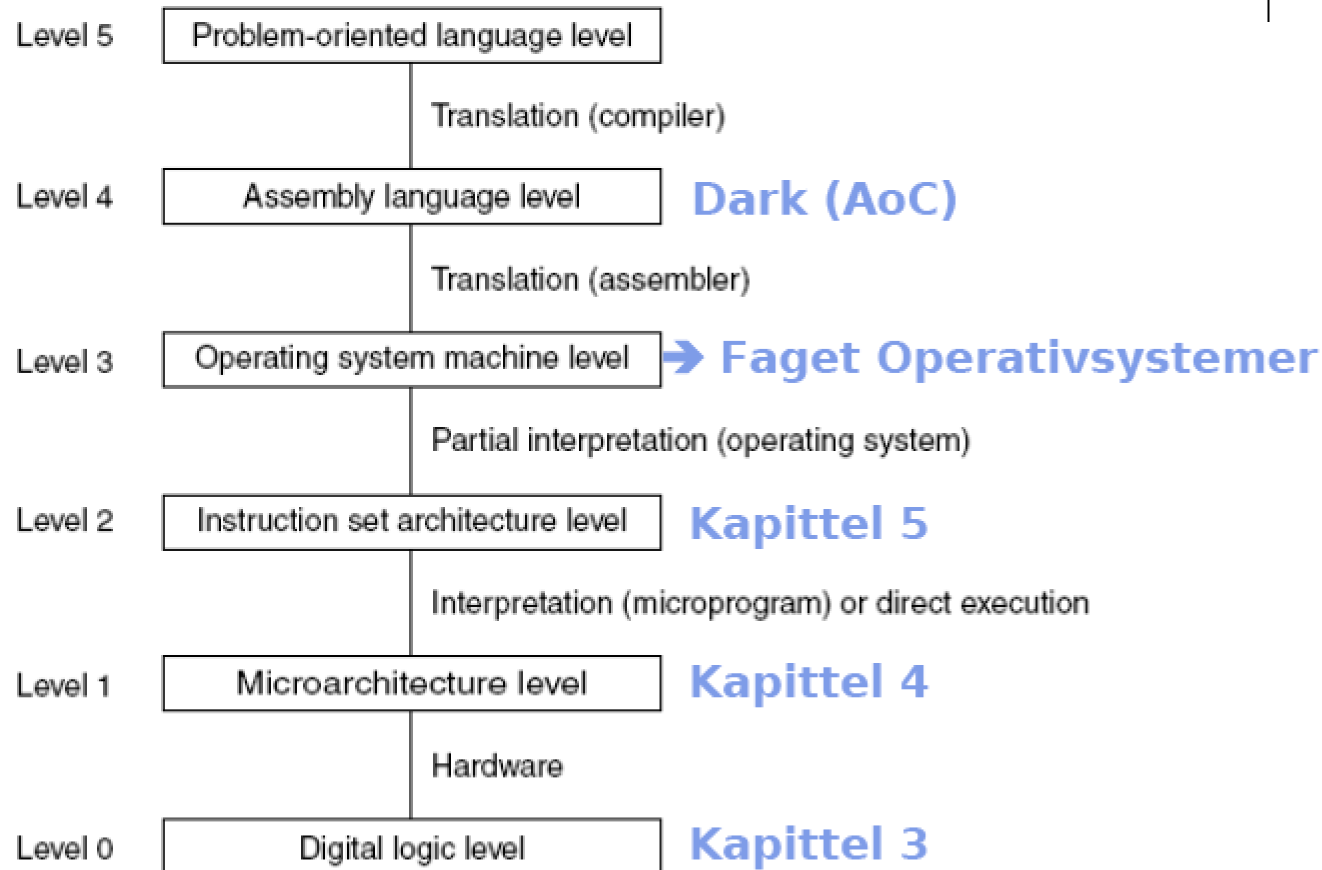
ISA Instruction Set Architecture (5)

NTNU
Innovation and Creativity

ISA



ISA vs. mikroarkitektur

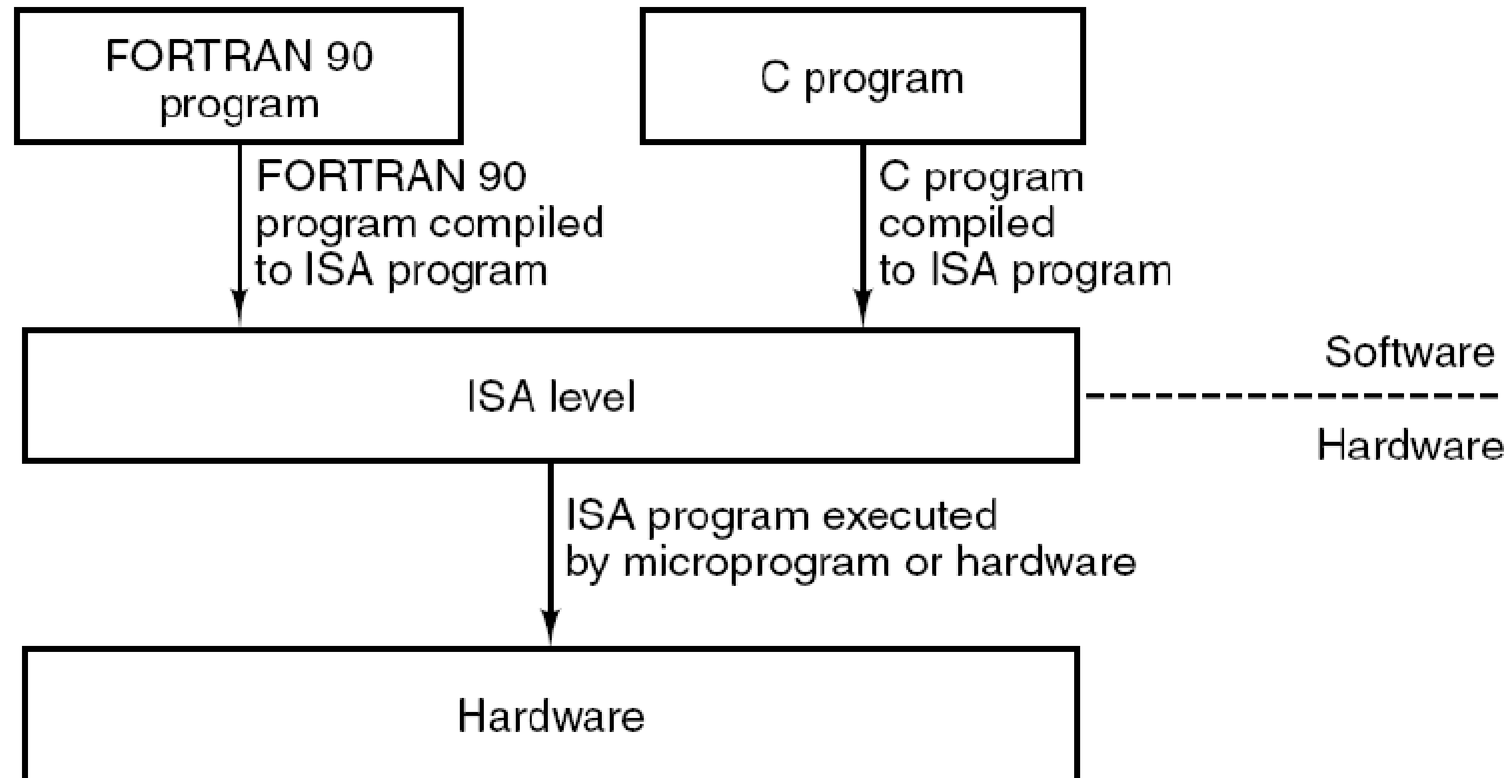
ILOAD j
ILOAD k
IADD
ISTORE i
ILOAD i
BIPUSH 3
…

ISA-instr.

NEXT_ADDRESS	JMPC	JAMN	JAMZ	SLL8	SRA1	F_0	F_1	ENA	ENB	INVA	INC	H	OPC	TOS	CPP	LV	SP	PC	MDR	MAR	WRITE	READ	FETCH	B bus
NEXT_ADDRESS	JMPC	JAMN	JAMZ	SLL8	SRA1	F_0	F_1	ENA	ENB	INVA	INC	H	OPC	TOS	CPP	LV	SP	PC	MDR	MAR	WRITE	READ	FETCH	B bus
NEXT_ADDRESS	JMPC	JAMN	JAMZ	SLL8	SRA1	F_0	F_1	ENA	ENB	INVA	INC	H	OPC	TOS	CPP	LV	SP	PC	MDR	MAR	WRITE	READ	FETCH	B bus

Mikroprogram: Sekvens av μInstr.

ISA



ISA

- Opprineleg det einaste nivået
  - Ikkje skilje (microArc, ISA), huks ledningar og brytarar som programmering. Programmering direkte på samankopling
  - Ofte kalla ”architectur”
    - Funksjonelle einingar
    - Samankopling
    - Gir kva instruksjonar som er mest anvendelege
  - Grense mellom maskinvare og programvare
    - Maskinvare dårleg egna til å utføre C/C++/JAVA code
    - Maskina kan utføre maskinkode (ISA)
    - Kompilerar til ISA-code
      - Kan optimalisera til mikroarkitektur (486 med utan flyttal eining)
  - Kan definer ISA lage forskjelige mikroarkitekturar
    - Bakoverkompatibel
    - Ved ny mikroarkitektur
      - Nye instruksjonar
      - Auka yting (pipline, superskalaritet…)

ISA

- Kva er ein bra ISA
  - Godt design gir auka yting
  - To målgrupper for ISA
    - Maskinvaredesignar
      - Implementere ein ISA effektivt (må vere effektiv for maskinvare)
    - Programvaredesignar
      - Enkelt å generere ISA-kode (kompilator)
        - Støtte datastrukturar
        - Støtte funksjonar (vanlege) effektivt
        - Må kunne kompilere program til ISA-kode som er effektiv (prøv å skrive C-prog for berekningar på IJVM, **NOT**)

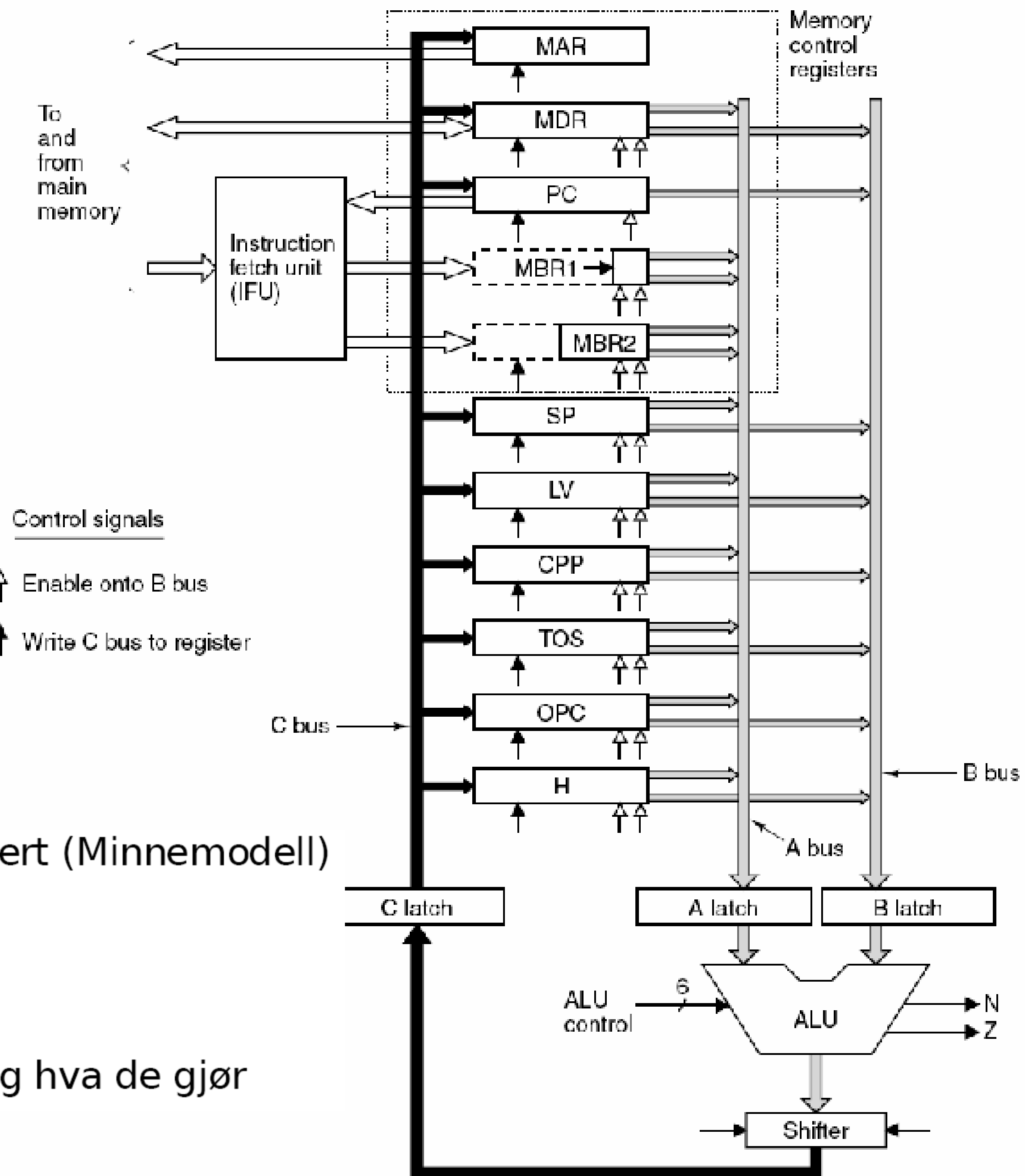
NTNU
Innovation and Creativity

ISA: kva er definert

- For å kunne skrive ISA-kode, må man vite:
  - Hvordan hovedlageret er organisert (Minnemodell)
  - Hvilke registre som finnes
  - Hva registrene skal brukes til
  - Hvilke datatyper som finnes
  - Hvilke instruksjoner som finnes og hva de gjør
  - osv...
- Svarene på alt dette definerer ISA-nivået

NTNU
Innovation and Creativity

ISA: kva er definert



- Hvordan hovedlageret er organisert (Minnemodell)
- Hvilke registre som finnes
- Hva registrene skal brukes til
- Hvilke datatyper som finnes
- Hvilke instruksjoner som finnes og hva de gjør

ISA: kva er definert

- I følge forrige foil, er trenger man f.eks. ikke å vite om det er samlebånd eller ikke i μArk
- Men: Kan ha konsekvenser for optimalisering
  - Eksempel: Superskalar med en heltallsenhet og en flyttallsenhet
  - Da optimalt med annenhver heltallsinstr. og flyttallsinstr. i ISA-koden
- Derfor: Kan være nødvendig å vite detaljer om μArk for å lage optimal ISA-kode

NTNU
Innovation and Creativity

ISA: Minnemodell

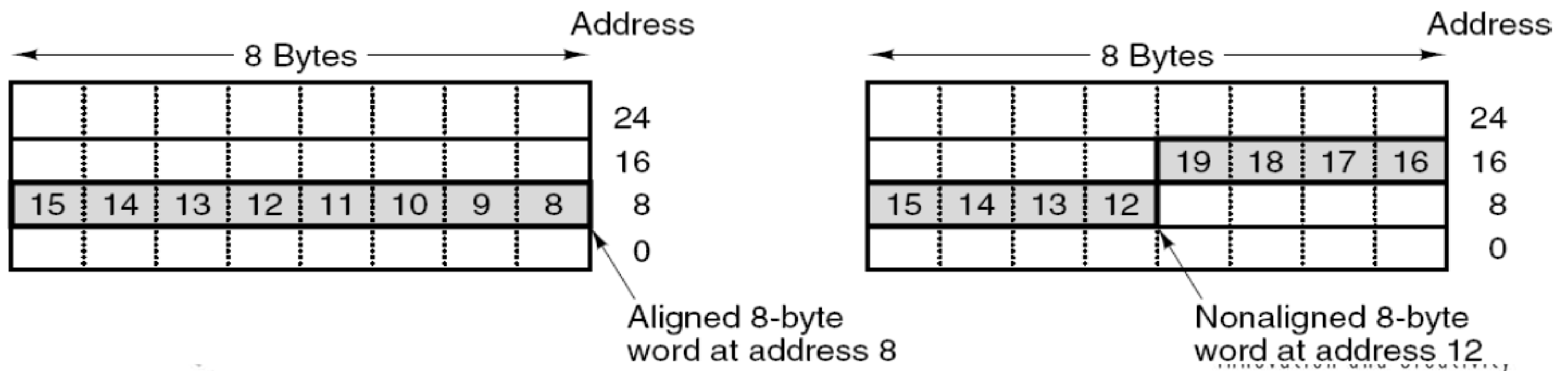
Omfatter bl.a.:

- Størrelse på adresser (antall bit)
  - Normalt 32 bit som gir 4 G adresser
- Adresserbar enhet
  - Normal 8 bit (byte), men 1-60 bits har eksistert
- Organisering i større enheter
  - Ord på typisk 4 byte (32 bit) eller 8 byte (64 bit)
  - Mange instruksjoner manipulerer ord om gangen

NTNU
Innovation and Creativity

”Alignment”

- Kan et 8 bytes ord ha vilkårlig adresse?
  - Nei → ”Alignment”. Kan kun ligge på adresse 0, 8, 16..
  - Ja → ”Nonalignment”
- Ofte enklere og raskere å hente ord som ligger på ”naturlige” adresser
- P4 kan ikke oppgi 3 siste adressebit, men må støtte vilkårlige adresser pga. bakoverkompatibilitet

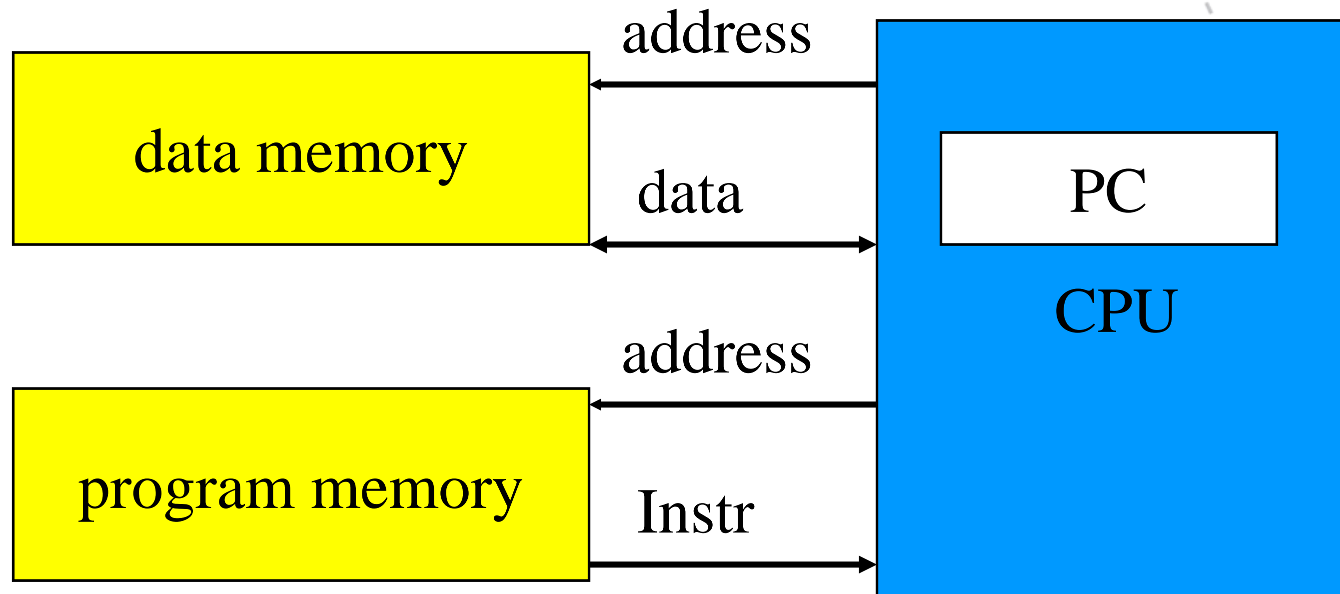


Adresserom

- Normalt har man kun ett adresserom – felles for data og instruksjoner
- Alternativ: Forskjellig adresserom
  - En instruksjon på adresse 8 og et dataelement på adresse 8 ligger ikke på samme sted
  - 32 bits adresse gir dermed $2x2^{32}$ lokasjoner
  - Skriveoperasjoner kan ikke ødelegge instruksjoner
- NB! Ikke det samme som delt hurtigbuffer
  - Kan ha delt hurtigbuffer med ett adresserom

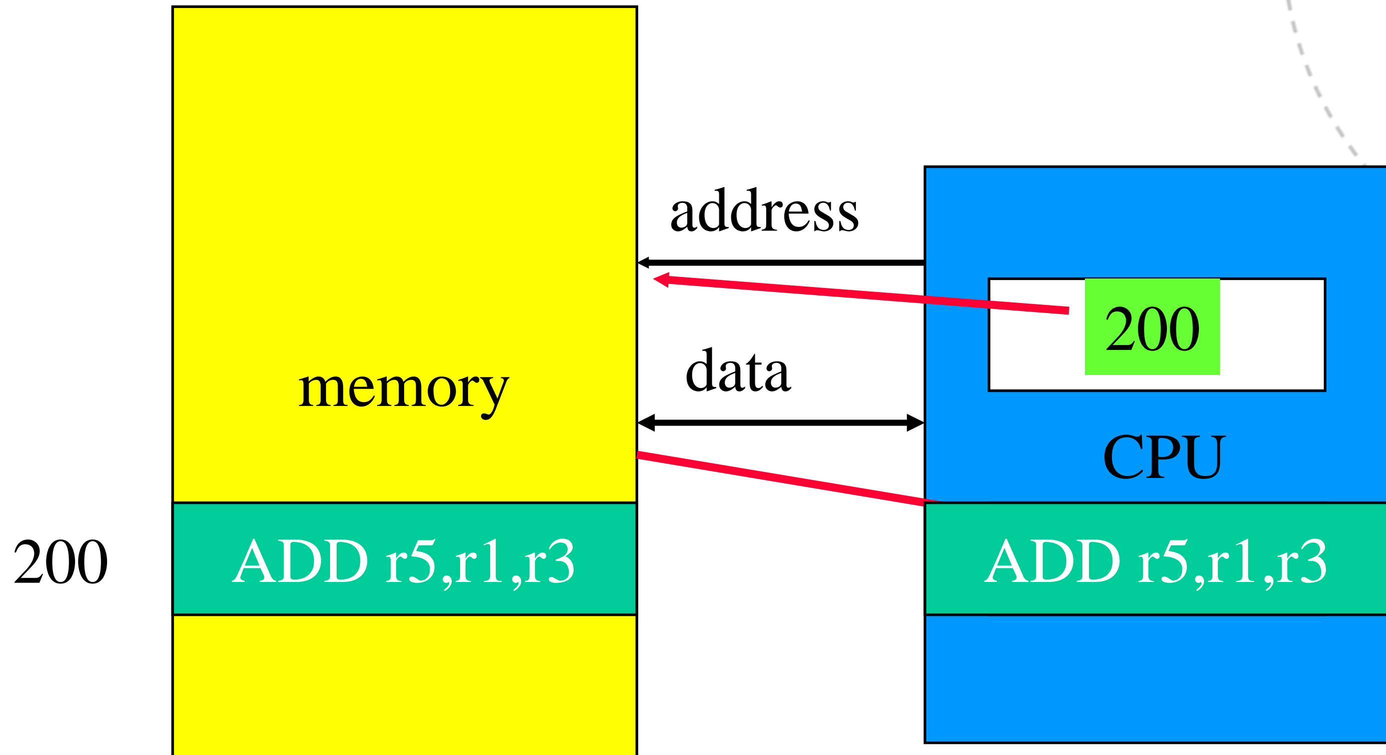
NTNU
Innovation and Creativity

Adresserom



Harvard architecture

Adresserom



von Neumann architecture computer

Register

- Registre er på toppen av minnehierarkiet
  - Viktig ressurs å utnytte effektivt
- Noen registre er skjult fra ISA-nivået
  - Eksempel: TOS og MAR fra Mic-eksempel i kap 4
- To typer
  - Generelle registre
    - Lokale variable, mellomlagring av resultater, ..
  - Spesialregistre
    - Programteller, stakkpeker
    - Statusord (PSW) – inneholder bl.a. ALU-flagg

Kjerne/brukar modus

- De fleste prosessorer har (minst) to modi på ISA-nivået: Kjernemodus og brukermodus
- Operativsystem kjører i kjernemodus, programmer i brukermodus
- Brukermodus begrenser hvilke instruksjoner som kan utføres og hvilke registre som kan leses fra / skrives til
  - Gjør det vanskeligere for et program å ødelegge for (el. spionere på) et annet

NTNU
Innovation and Creativity

Instruksjonar

- Hvilke instruksjoner som finnes og hvordan de fungerer er helt sentralt i ISA-nivået
- Viktigste typer:
  - Flytting av data
  - Aritmetiske operasjoner
  - Sammenligninger og hoppinstruksjoner
  - Prosedyrekall
  - Løkker
  - I/O

NTNU
Innovation and Creativity

ISA: eksempel

- Pentium 4 (IA-32)
- 8051

- Fokus på
  - Historikk & bakgrunn
  - Egenskaper

Pentium 4

- Brukes i «desktop»-maskiner, «server-farms»
- 80386 og nyere Intel-prosessorer har i all hovedsak hatt samme ISA-arkitektur: IA-32
  - Tidligere prosessorer var 4, 8 eller 16-bit
  - Hovedendringer siden 80386 har vært nye instruksjoner: MMX, SSE, SSE2
- Pentium 4 kan likevel utføre programmer skrevet for prosessorer helt tilbake til 8088
  - Real mode, Virtual 8086 mode

Pentium 4 IA-32 eigenskapar

- Har kjernemodus og brukermodus (+ to til)
- Adresserom delt i 16 386 segmenter, hvert segment med adresser fra 0 til 2^{32}
  - I praksis brukes bare 1 segment
- Få/ingen helt generelle registre
  - Alle har sitt spesialformål
  - De fleste finnes i 8, 16 og 32-bits versjoner
  - Dermed vanskelig å lage god kompilator

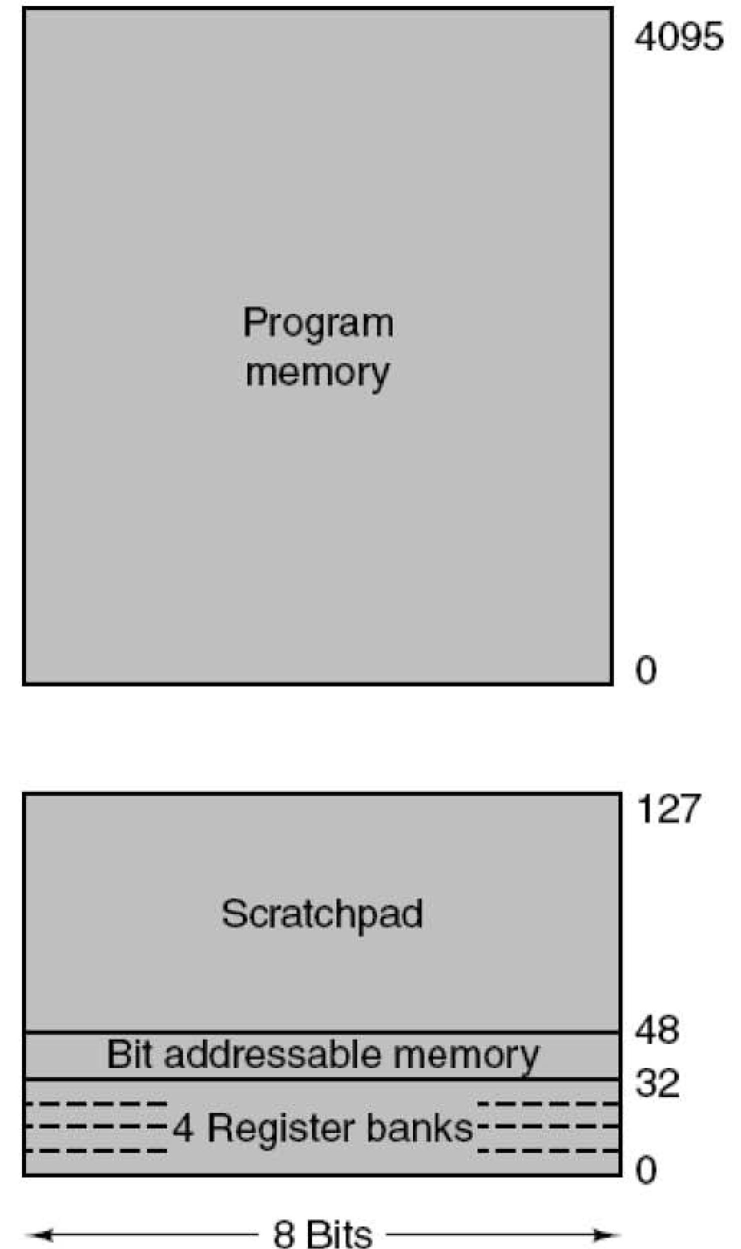
NTNU
Innovation and Creativity

8051

- Brukes ofte i «embedded» systemer
- Basert på Intels 8080 – laget for å ha alt på en chip
- Gammelt design, men enkelt og billig
- Finnes i flere varianter med varierende størrelse på lager
- I hovedsak kun lager-på-brikke, men mulighet for aksess av ekstern lager
- Ikke separate kjerne-/brukermodi
  - Kjører aldri flere program samtidig

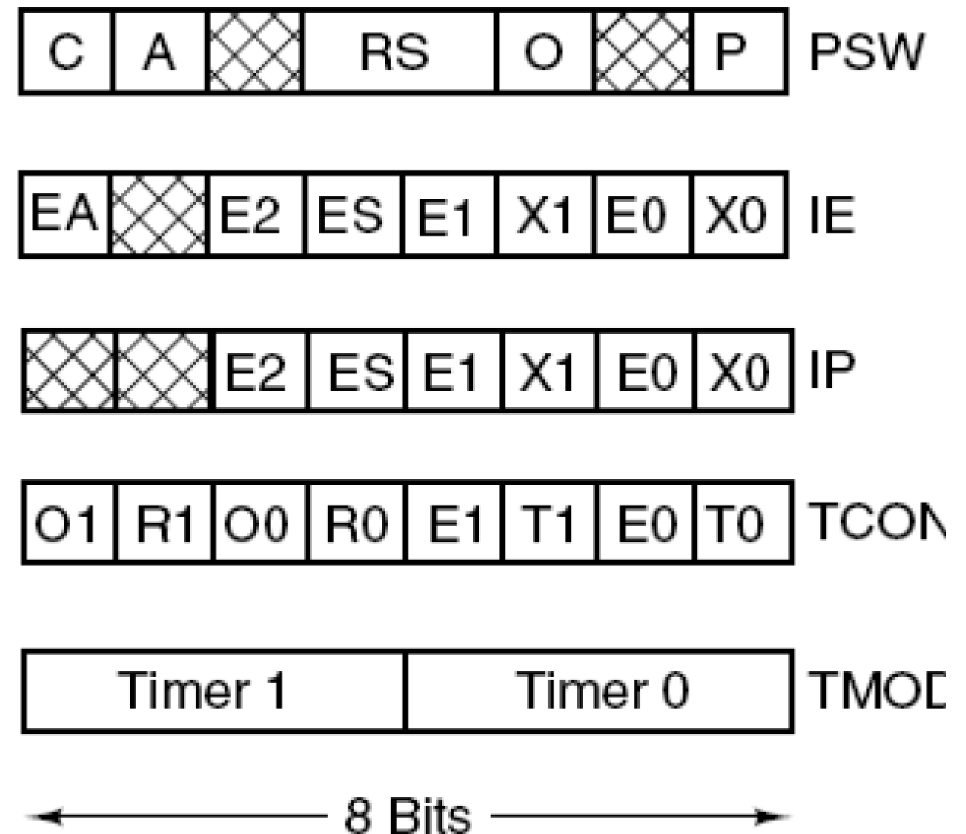
8051: Minnemodell

- Separate adresserom for program og data
  - Dermed kan program være i ROM og data i RAM
- 4 sett av 8 8-bits registre
  - Ligger først i data-adresserom
  - Bytter mellom sett under avbruddshåndtering
  - Gir veldig rask bytting, bra for mye I/O og sanntidssystemer
- 16 bytes bitadresserbart minne
  - Praktisk for boolske variable
  - Spesial-instruksjoner



8051 register bruk

- PSW: Statusord
  - RS – aktivt registersett
- IE – avbruddskontroll
  - Eksterne avbrudd
  - Avbrudd pga. timere
- IP – avbruddsprioritet
  - To nivå: Høy/lav
- TCON – timerkontroll
- TMOD – timermodus
  - Timer eller teller



NTNU
Innovation and Creativity

Datatypar

- En datamaskin behandler data
  - Heltall, flyttall, ikke-numeriske data
- Maskinvarestøtte for datatype
  - ISA-Instruksjoner som opererer på spesifikke format (eks: adderer to 32-bits 2s-kompl. heltall)
  - Spesifisert hvordan data skal lagres binært
- Alternativ: Programvarestøtte
  - Eks: Programmerer lager rutine for håndtering av 64-bits heltall vha. to 32-bits heltall

Heiltal

- Som regel maskinvarestøtte for flere størrelser (eks: 8, 16, 32, 64 bit)
- Med eller uten fortegn
  - Uten fortegn gir større maksimaltall
- Negative tall håndteres som regel vha toerskomplement (antas kjent – se app A)
- BCD: Binary Coded Decimal
  - Spesialformat for desimaltall (titallssystemet)
  - 4 bit lagrer hvert siffer

NTNU
Innovation and Creativity

Flyttal

- Så langt: Heltall på binær form
  - Problem 1: Hva med 3,14?
- 32 bit kan lagre 2^{32} ulike tall
  - Eks: Positive heltall 0 – 4 294 967 295
  - Problem 2: Hva da med større/mindre tall?
- Løsning: Flyttall
  - $\pm F * B^{\pm E}$ (Fraksjon/Mantisse, Base, Eksponent)
  - $3{,}14 = 314 * 10^{-2}$ ($= 3140 * 10^{-3}$)
  - $5\ 000\ 000\ 000 = 5 * 10^{9}$

NTNU
Innovation and Creativity

Ikkje numeriske datatypar

- Karakterer (tegn)
  - ASCII (7 bit) eller Unicode (16 bit)
- Boolske verdier
  - Som regel brukes 1 byte per verdi
  - Unntak: Bitmap med boolske verdier
- Peker
  - Adresser til lagerlokasjoner (typisk hovedlager)
- Multimedia (Eks: fargeverdier)

Datatypar eksempel

Pentium 4

Type	1 Bit	8 Bits	16 Bits	32 Bits	64 Bits	128 Bits
Bit						
Signed integer		×	×	×		
Unsigned integer		×	×	×		
Binary coded decimal integer		×				
Floating point				×	×	

UltraSPARC III

Type	1 Bit	8 Bits	16 Bits	32 Bits	64 Bits	128 Bits
Bit						
Signed integer		×	×	×	×	
Unsigned integer		×	×	×	×	
Binary coded decimal integer						
Floating point				×	×	×

8051

Type	1 Bit	8 Bits	16 Bits	32 Bits	64 Bits	128 Bits
Bit	×					
Signed integer		×				
Unsigned integer						
Binary coded decimal integer						
Floating point						

Instruksjonsformat

- Assembler: ADD R1, R2, R3
- Maskinkode: 101010111001011010100....
- Instruksjonsformatet forteller hvilke bits som tilsvarer hva
- Generelt: Opkode + Adresser til operander
  - Hvor mange operander er eksplisitt?
    - Implisitt: Gitt av opkode eller annen operand
  - Operander kan være registre eller hovedlageradr.
    - Hvor mange kan være hovedlageradresser?

NTNU
Innovation and Creativity

Explisite operandar

- Gitt C = A + B
  - 3 ”operander” – A, B og C
  - Hvor mange skal kunne oppgis eksplisitt?
- Viktig avgjørelse mhp. arkitektur
  - Mange eksplisitt ➔ enklere kode
  - Få eksplisitt ➔ kortere instruksjoner
- Gjennomgående eksempel:
  - X = (A + B) * (C + D)
  - Alt er hovedlageradresser
  - Innholdet i A, B, C, D skal ikke endres

3-adresseinstruksjoner

X = (A + B) * (C + D)

- ADD T1, A, B
- ADD T2, C, D
- MUL X, T1, T2

- T1 og T2 brukes til mellomlagring (kan være i hovedlager eller registre)
- Hvis ADDR1-3 er hovedlageradr., trenger vi ikke registre

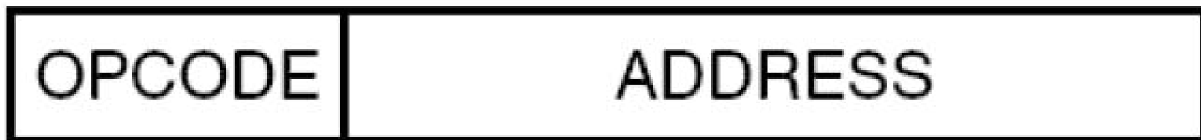
OPCODE	ADDRESS1	ADDRESS2

2-adresseinstruksjoner

X = (A + B) * (C + D)

- MOVE T1, A
- ADD T1, B
- MOVE X, C
- ADD X, D
- MUL X, T1

- Implisitt: Første operand er også der svar havner
- Må bruke T1 for å ikke ødelegge innholdet i A



1-adresseinstruksjoner

X = (A + B) * (C + D)

- LOAD A
- ADD B
- STORE X
- LOAD C
- ADD D
- MUL X
- STORE X

- Implisitt – akkumulator. Kan både være en operand og der svar havner
- LOAD A
  - ACC ← M[A]
- ADD B
  - ACC ← ACC + M[B]
- STORE X
  - M[X] ← ACC

OPCODE

0-adresseinstruksjoner

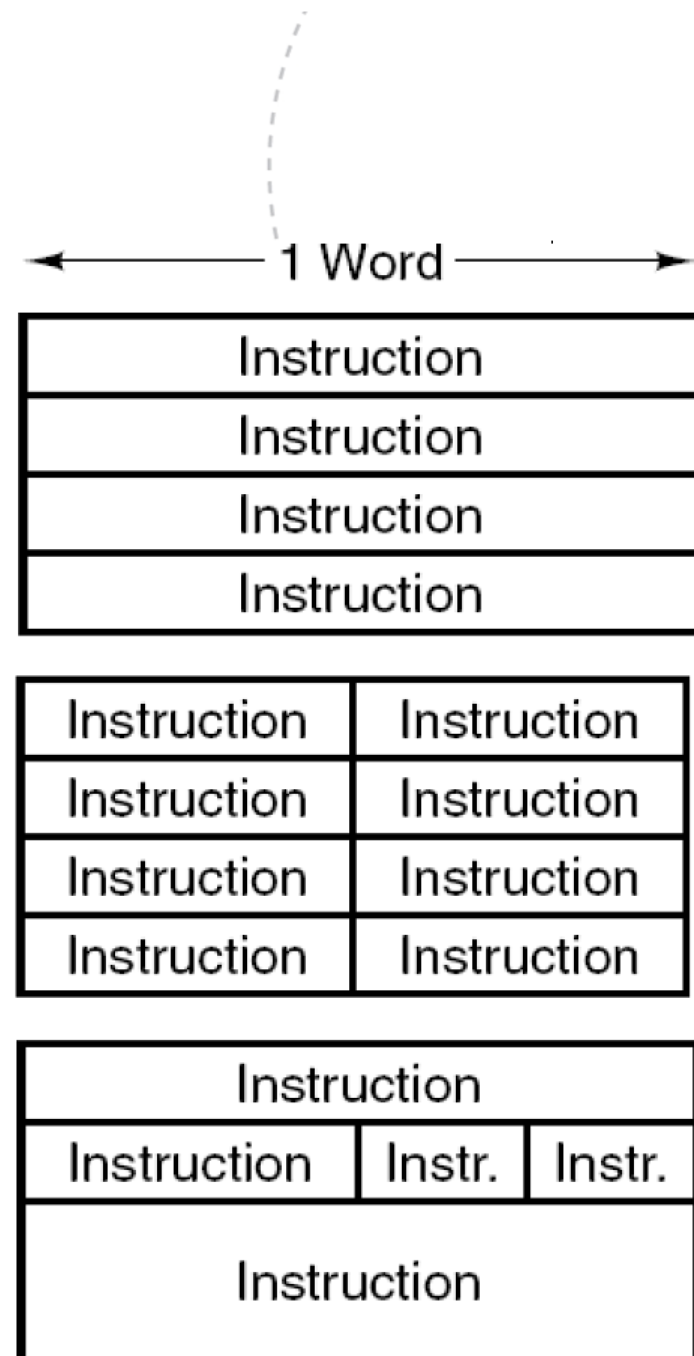
X = (A + B) * (C + D)

- PUSH A
- PUSH B
- ADD
- PUSH C
- PUSH D
- ADD
- MUL
- POP X

- Implisitt – alt. Bruker stakk for operander og svar
- Unntak Push & Pop

Instruksjonslengde

- Fast instruksjonslengde
  - Enklere dekoding
  - Fordel for samlebånd (spesielt superskalare)
  - Men: mulighet for sløsing med plassen
- Variabel lengde vanlig før, fast lengde vanlig i nye ISA

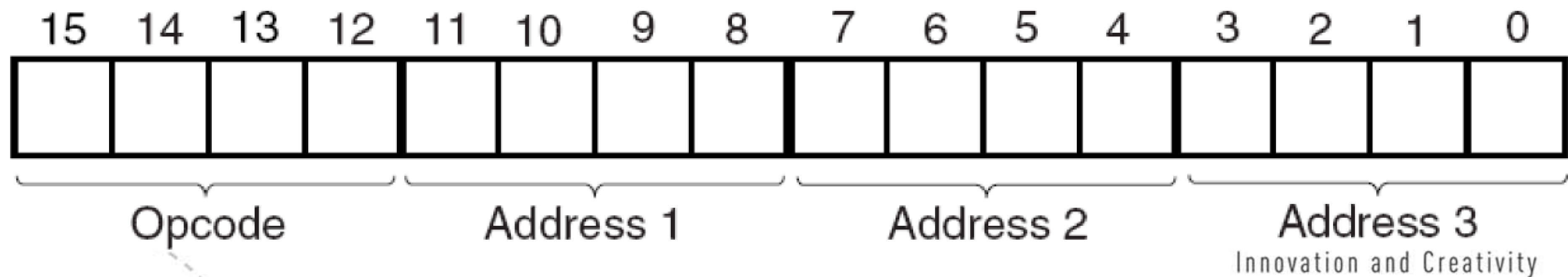


Instruksjonsformat kort/langt

- Opkodefelt
  - Langt felt gir mulighet for mange forskjellige instruksjoner og plass til å vokse på
- # adressefelt – diskutert tidligere
- Adressefelt
  - Langt felt gir mulighet for stort hovedlager, mange registre
- Overordnet
  - Bør være kortest mulig mhp. henting av instr.
  - For kort gir kompleks dekoding, mange begrensninger

Variabel instruksjonslengde

- Anta 16 bits instruksjonslengde og formatet under
- I utgangspunkt maks 16 forskjellige instruksjoner
- Men hva med instruksjoner som har færre enn 3 operander?
  - Opcode = 1111 kan bety instr. med 2 operander
  - Da kan "Address 1" brukes til å spesifisere 16 2-adresse instruksjoner
- Tilsvarende for 1- og 0-adresse instruksjoner



Variabel instruksjonslengde

```
0000 xxxx yyyy zzzz
0001 xxxx yyyy zzzz
0010 xxxx yyyy zzzz
 ⋮
1100 xxxx yyyy zzzz
1101 xxxx yyyy zzzz
1110 xxxx yyyy zzzz
```

15 3-address instructions

```
1111 0000 yyyy zzzz
1111 0001 yyyy zzzz
1111 0010 yyyy zzzz
 ⋮
1111 1011 yyyy zzzz
1111 1100 yyyy zzzz
1111 1101 yyyy zzzz
```

14 2-address instructions

```
1111 1110 0000 zzzz
1111 1110 0001 zzzz
 ⋮
1111 1110 1110 zzzz
1111 1110 1111 zzzz
1111 1111 0000 zzzz
1111 1111 0001 zzzz
 ⋮
1111 1111 1101 zzzz
1111 1111 1110 zzzz
```

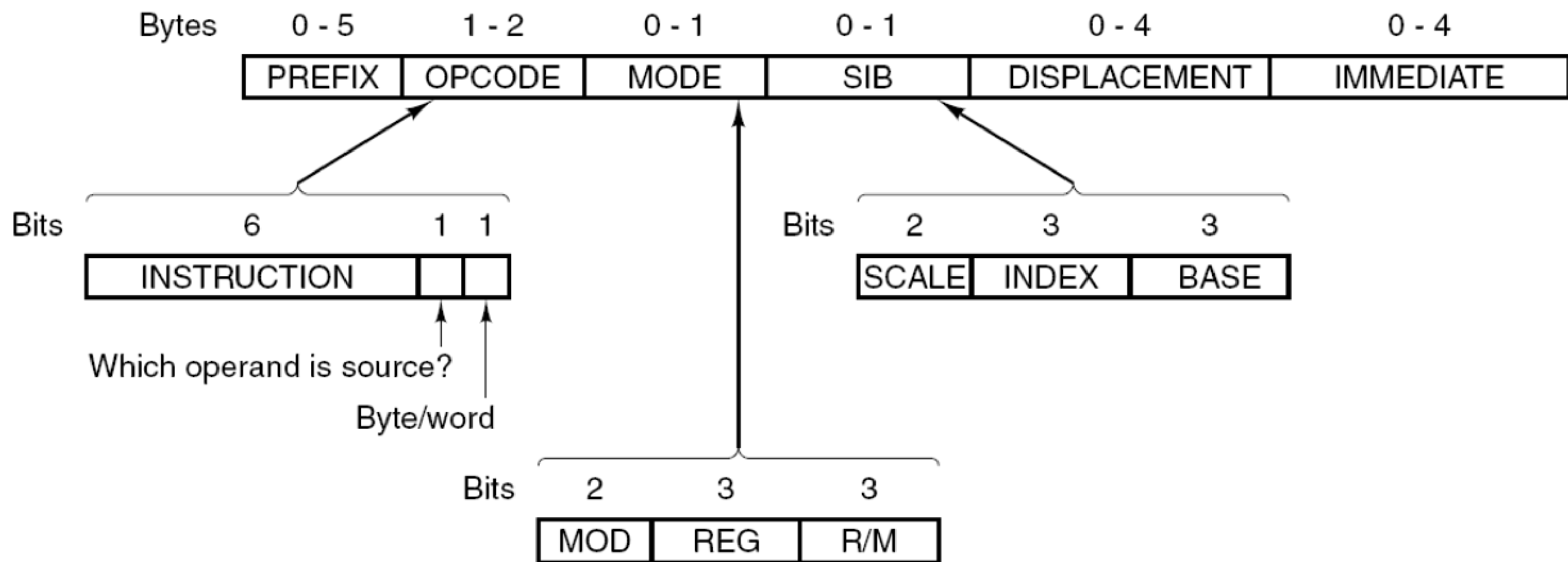
31 1-address instructions

```
1111 1111 1111 0000
1111 1111 1111 0001
1111 1111 1111 0010
 ⋮
1111 1111 1111 1101
1111 1111 1111 1110
1111 1111 1111 1111
```

16 0-address instructions

P4

- Komplekst og fullt av spesialtilfeller
- 6 felt av variabel lengde, kun 1 er obligatiorisk
- Mye 2-adresseinstruksjoner der 0-1 er hovedlageradr.
- Prefiks lagt til når Opcode ble for lite



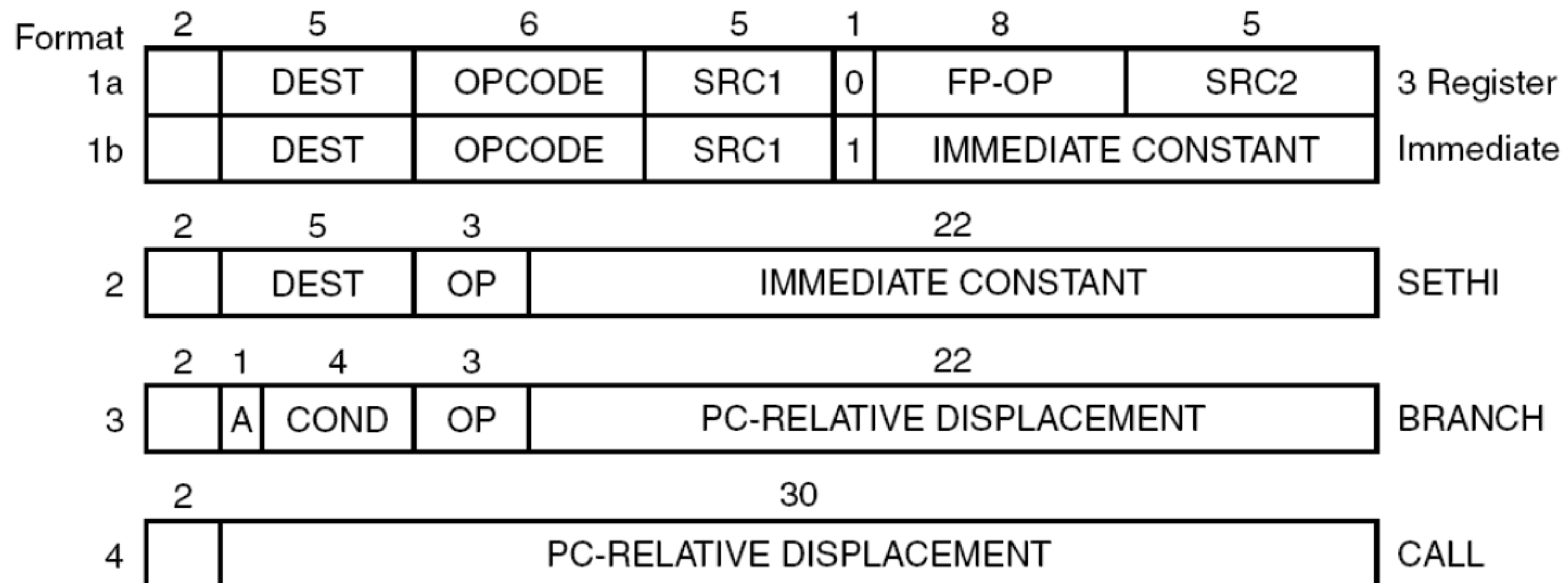
UltraSparc III

- Fast lengde, 32-bits

Som regel 3-adresseinstruksjoner, alt er registre

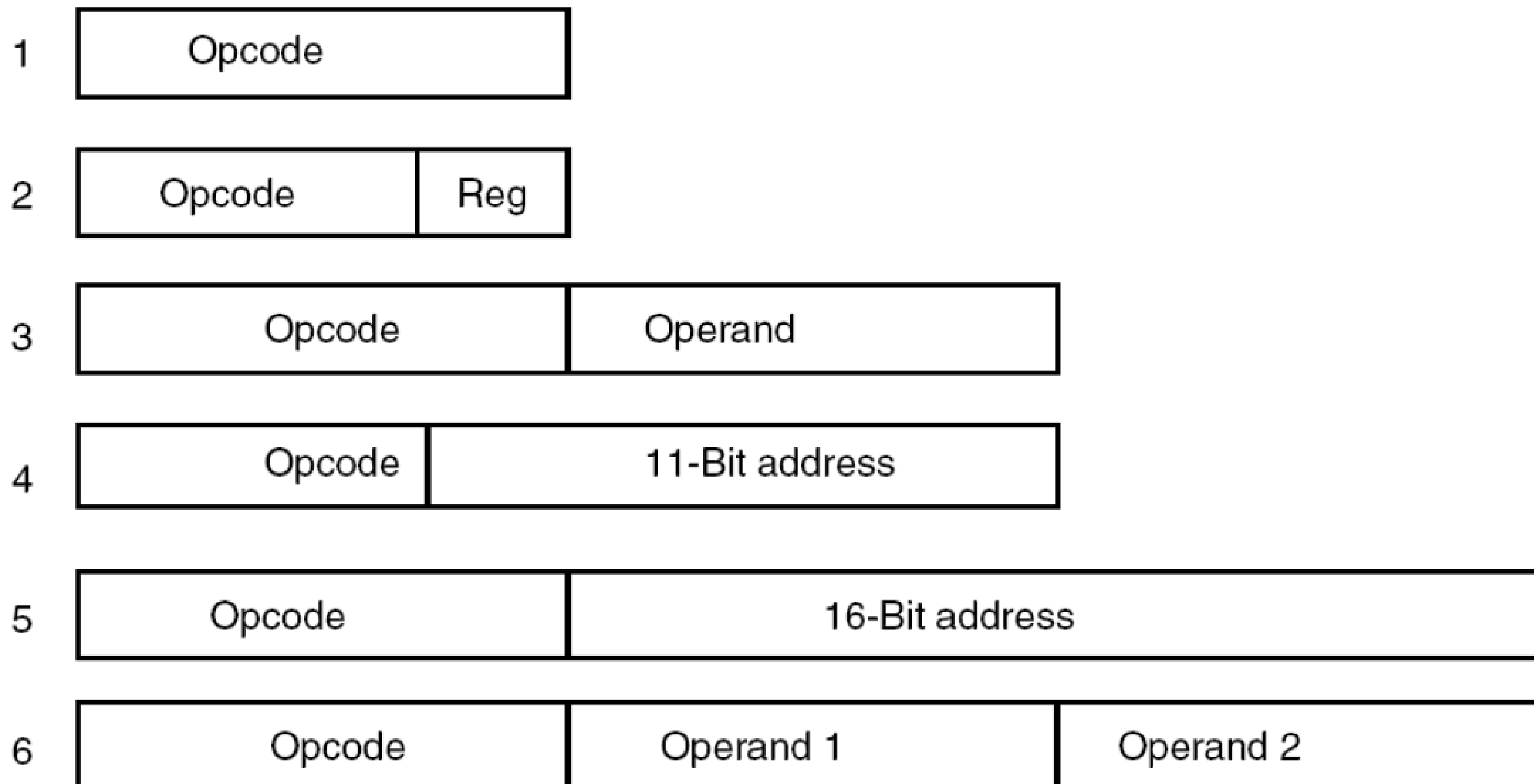
I utgangspunkt kun 4 format, flere lagt til senere

- 2 første bits forteller hvilket av disse 4 det er



8051

- Variabel lengde (plassbruk viktig, ikke samlebånd)
- Bruker akkumulator mye (1-adresseinstruksjoner)



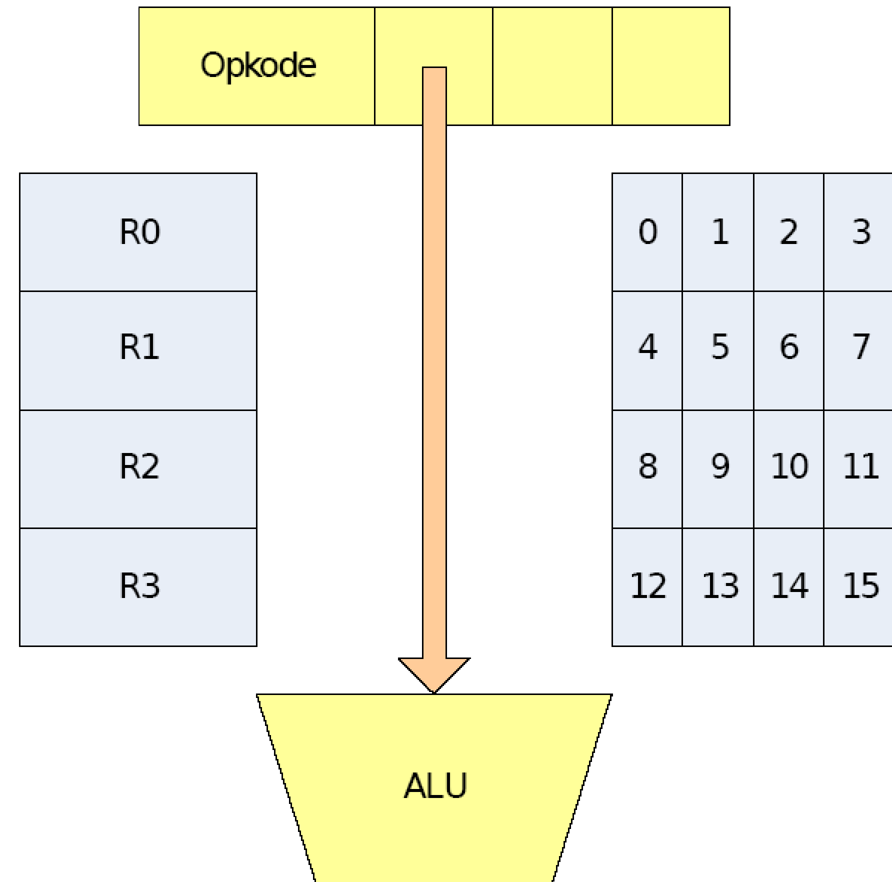
Adresseringsmodi

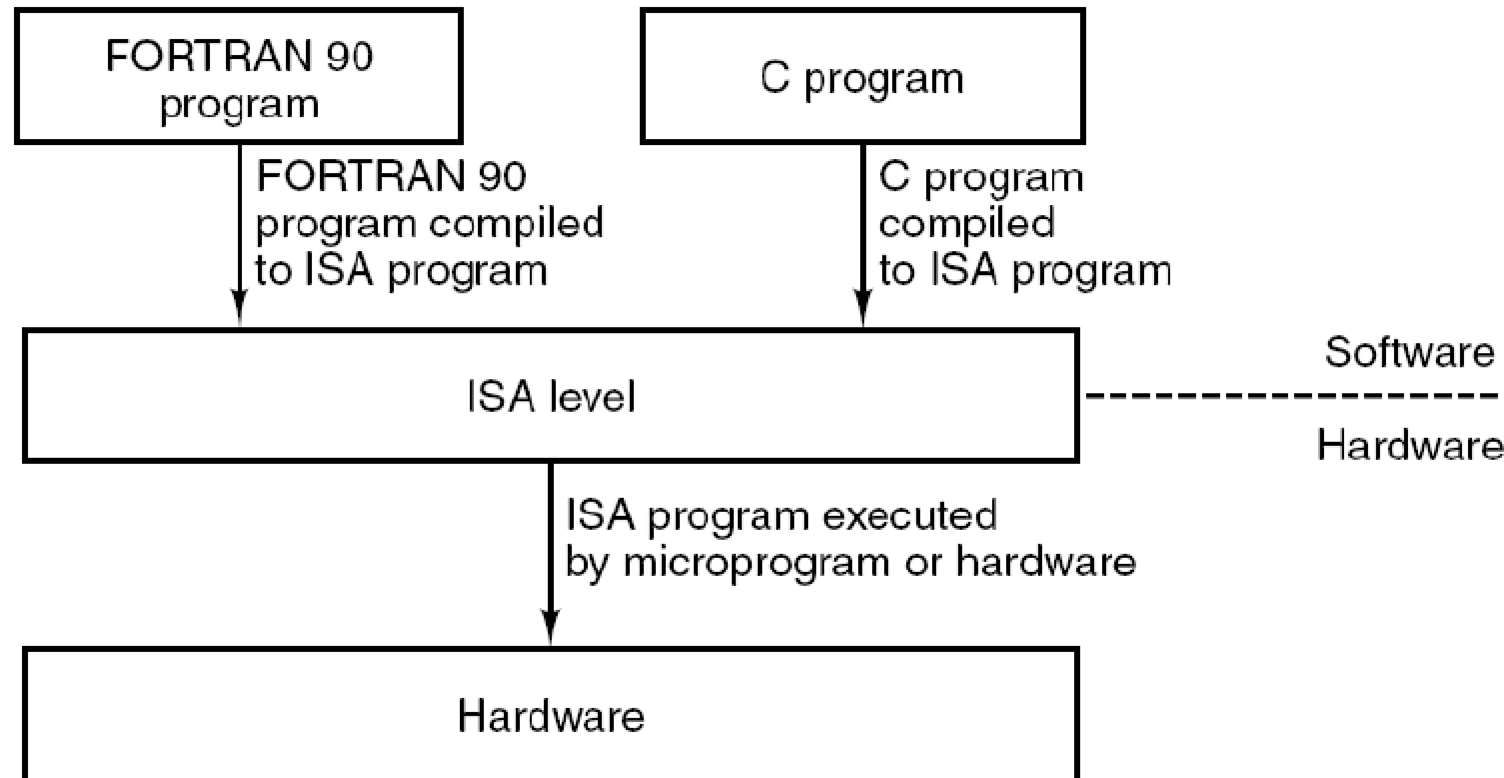
- Hvordan skal vi oppgi en operand?
- Adresseringsmodus
  - Regel for tolking av adressefelt
  - Mål: Effektiv adresse ➔ Peker på operand
  - Spesifiseres av opkode eller eget modus-felt i instruksjonen
  - Forskjellige operander i samme instruksjon kan ha forskjellig adresseringsmodus

NTNU
Innovation and Creativity

”Immediate” adressering

- Operanden ligger direkte i instruksjonen
- Er dermed tilgjengelig uten videre
- Størrelse på operand begrenset av feltlengde
- Kan bare brukes til konstanter – verdi bestemmes ved kompilering

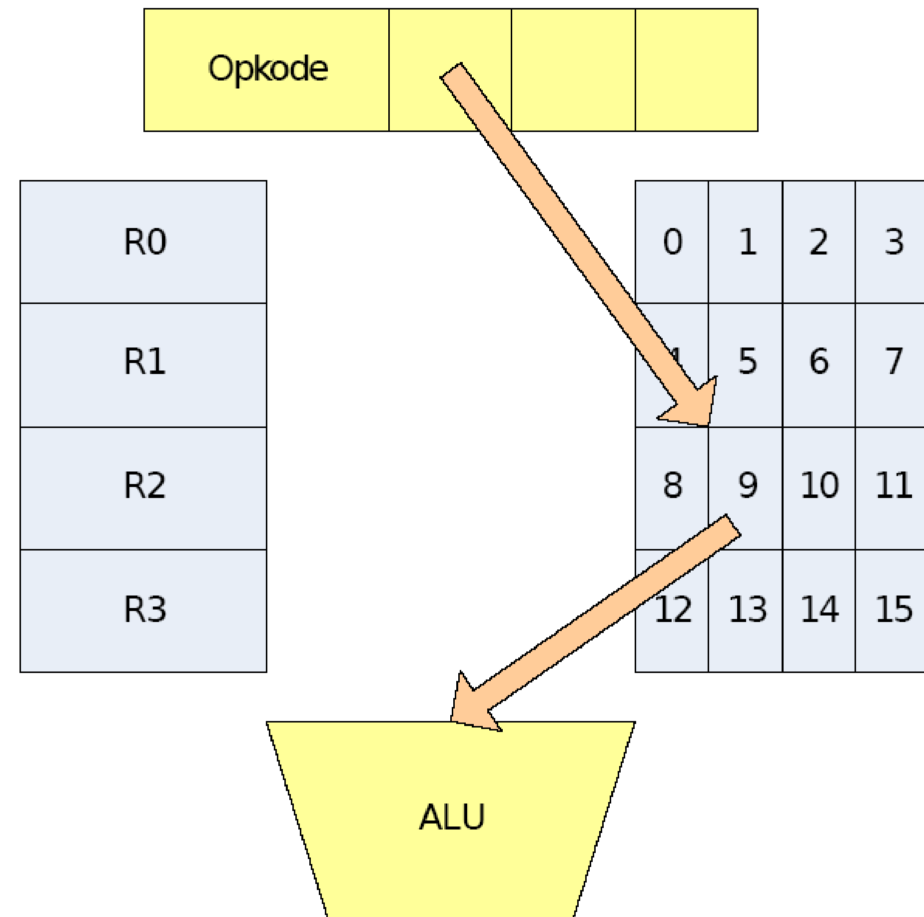




NTNU
Innovation and Creativity

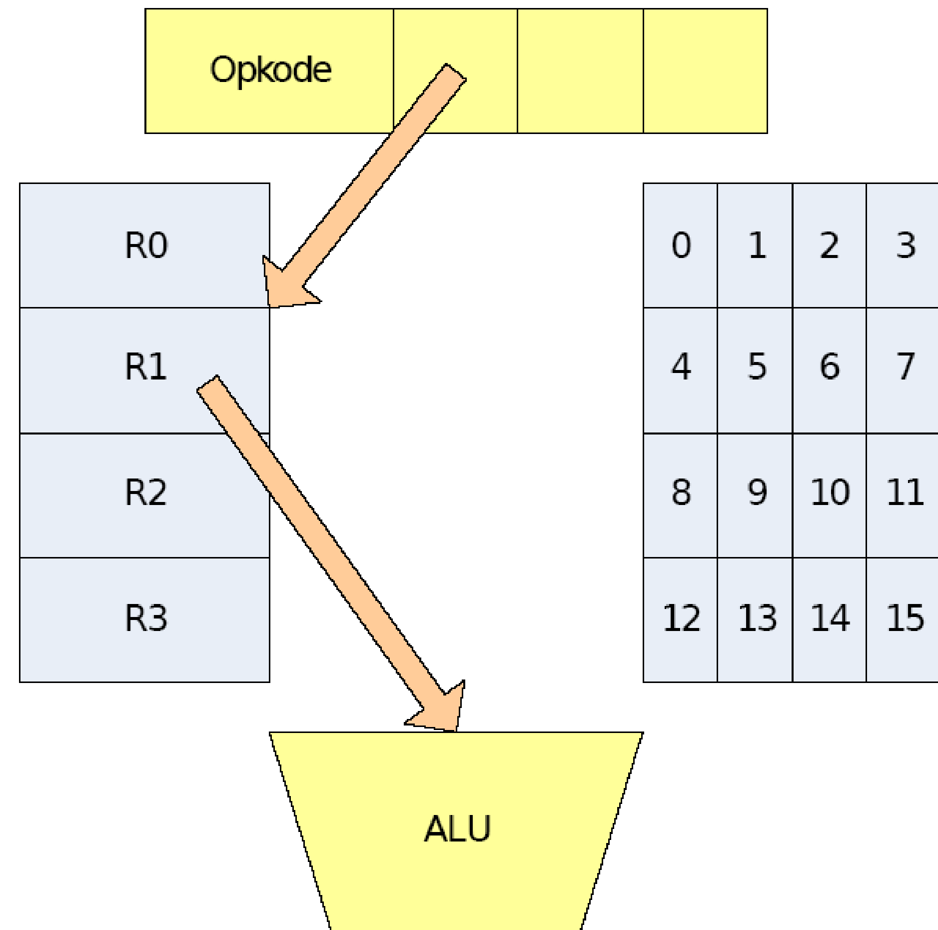
Direkte adressering

- Instruksjonsfelt inneholder hovedlageradresse
- Feltlengde begrenser adresseområde
- Adresse bestemt ved kompilering – lite fleksibelt
- Kan f.eks. bli brukt for globale variable



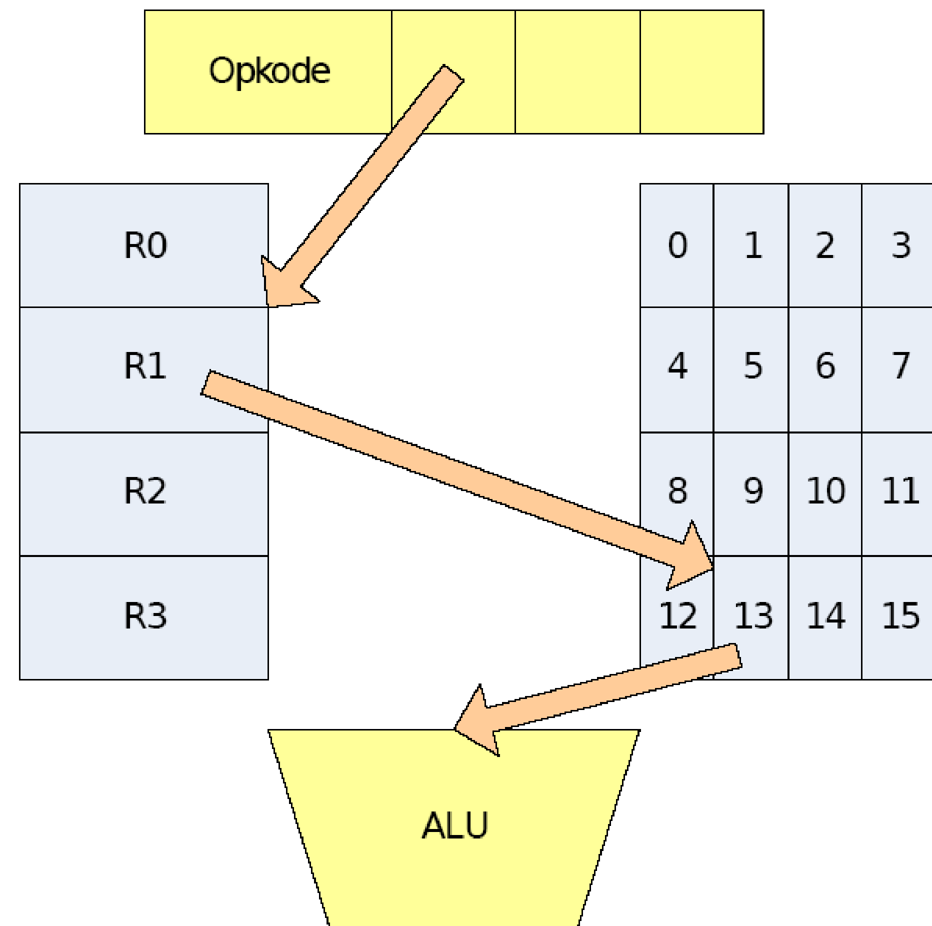
Registeradressering

- Instruksjonsfelt inneholder registernummer
- Veldig mye brukt
- RISC-ark. bruker nesten bare registeradressering
- Feltlengde begrenser antall registre



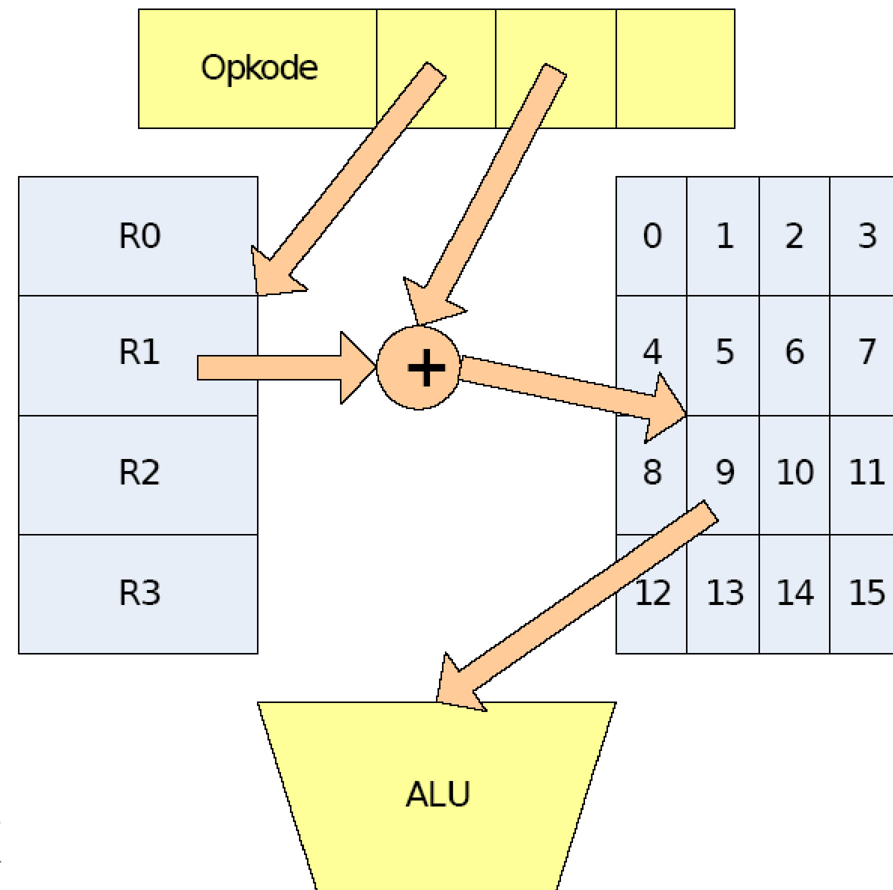
Register-indirekte adressering

- Instruksjonsfelt inneholder registernummer
- Register inneholder hovedlageradresse
  - Kalles *peker*
- Registernummer krever færre bit enn hovedlageradresse
- Fleksibelt



Indeksert adressering

- Instruksjonsfelt inneholder registernummer
- Register inneholder adresse1
- Instruksjonsfelt inneholder adresse2
- adresse1 + adresse2 = hovedlageradresse
- Tilsvarende som for lokale variable i Mic-ark (LV + variabelnummer)



Baseindeksert adressering

- Instruksjon inneholder to registernummer
- Hovedlageradresse er summen av innholdet i disse to registrene
- Kan ha offset i tillegg
- Eksempel
  MOV R4,(R2+R5)