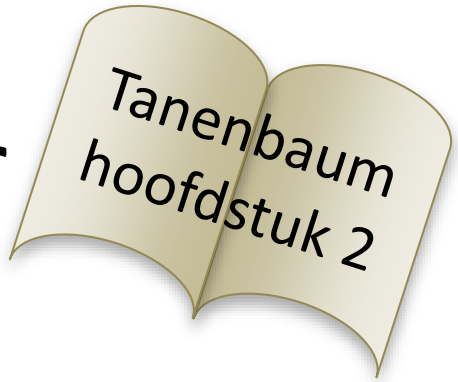
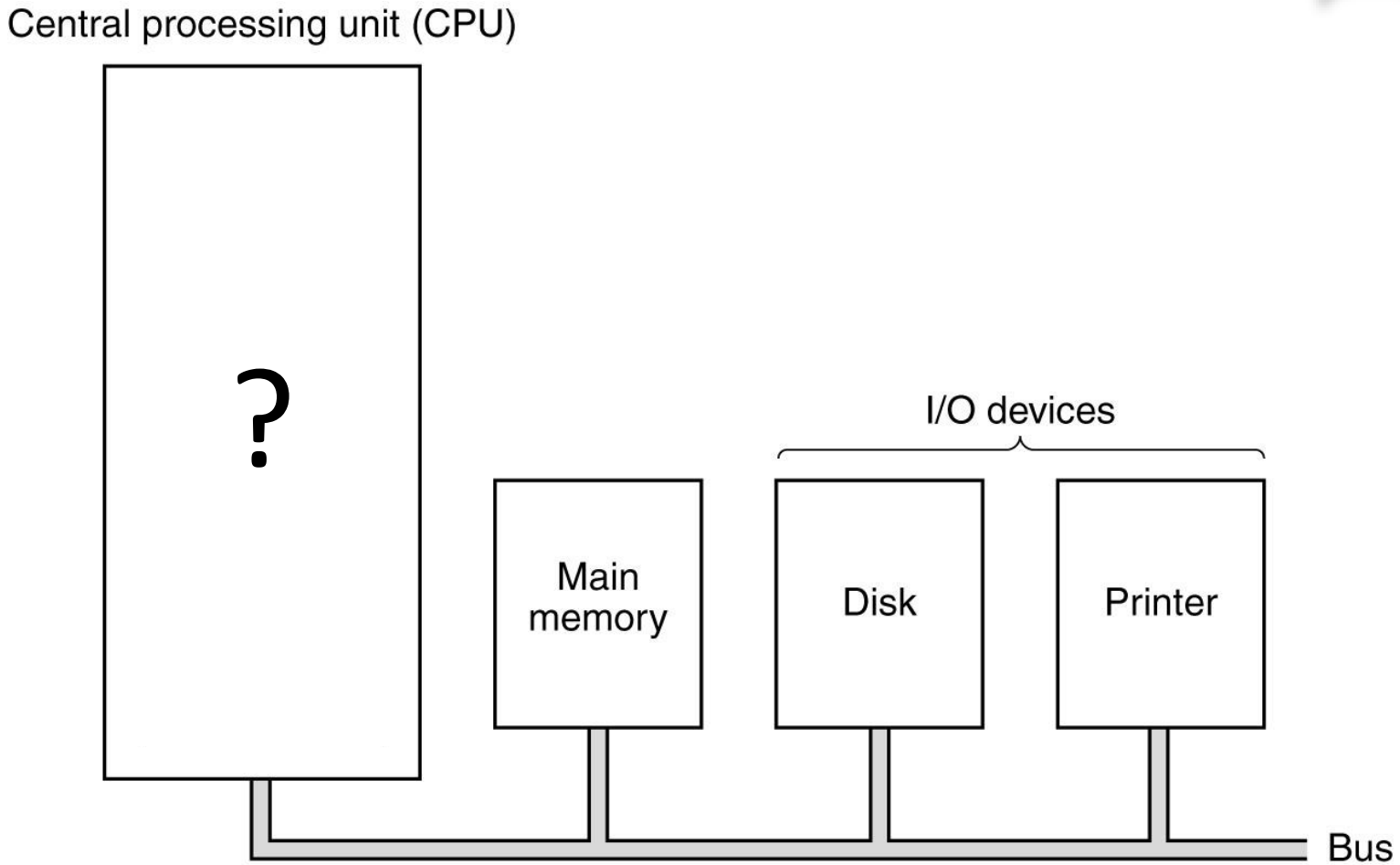
De CPU in detail
Hoe worden instructies uitgevoerd?

Processoren

8 december 2014

„von Neumann”-architectuur





Structuur van de CPU

- registers
  - gegevens- en adresregisters
  - vlaggen
  - instructie/programma-teller
  - interne registers
- arithmetisch-logische eenheid = ALU
- besturingseenheid = control unit

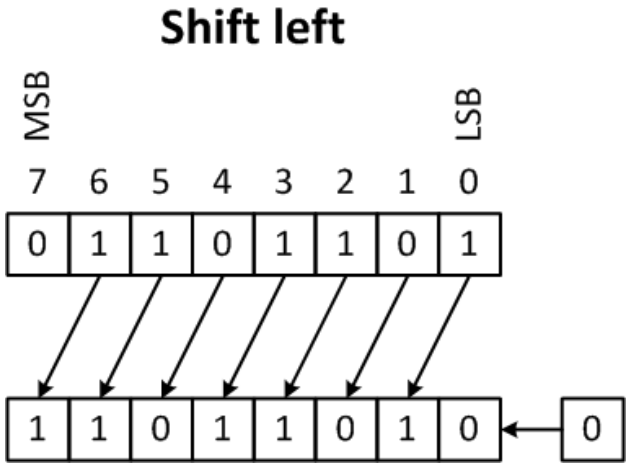
De ALU in detail

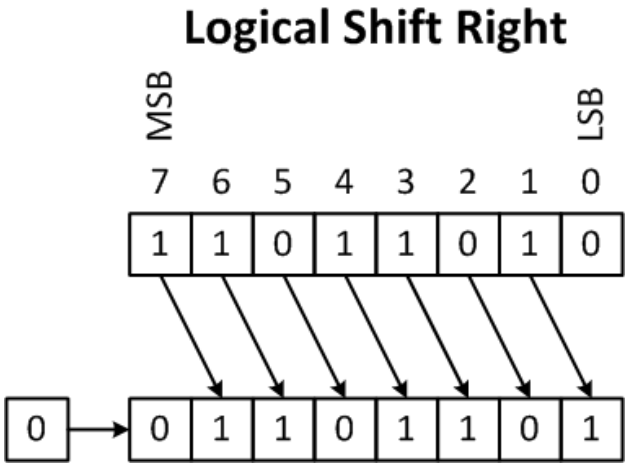
- Welke operaties moet de ALU aankunnen?
  - rekenoperaties: +, – (×, ÷)
  - logische operaties: AND, OR, NOT, XOR
  - bitshifts en bitrotaties

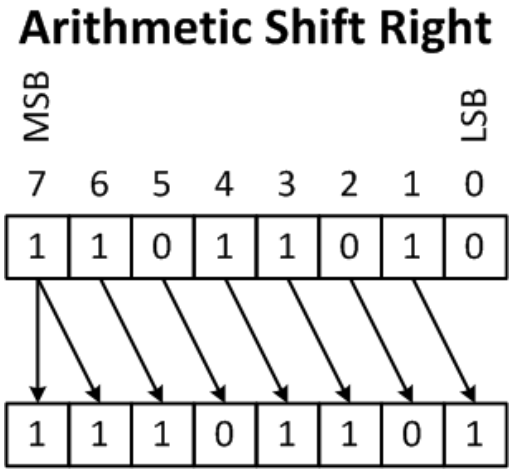
- efficiënte implementatie van aftrekking:

$$A - B = A + (-B) = A + (\text{NOT } B) + 1$$

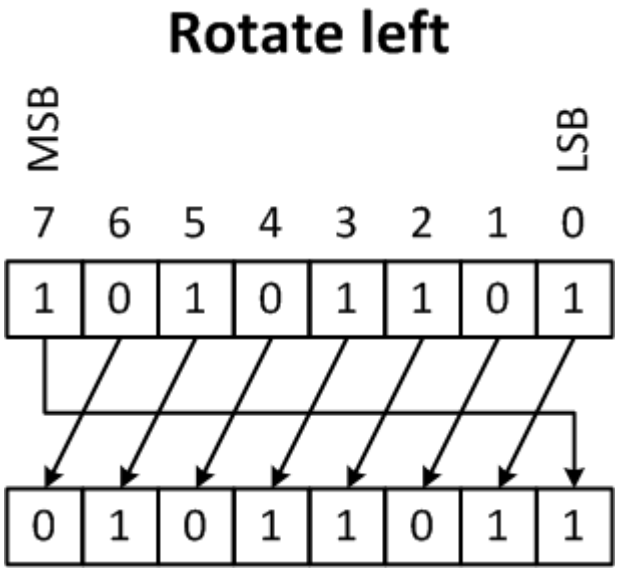
Bitshifts

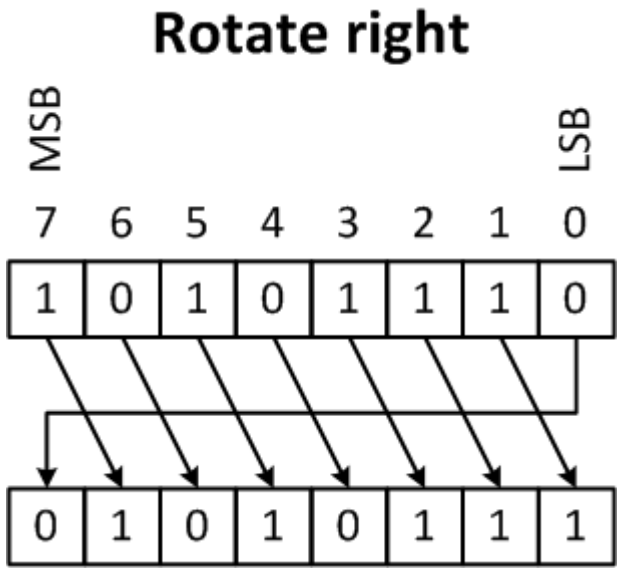

Shift left
MSB
LSB
7 6 5 4 3 2 1 0
0 1 1 0 1 1 0 1
1 1 0 1 1 0 1 0 0


Logical Shift Right
MSB
LSB
7 6 5 4 3 2 1 0
1 1 0 1 1 0 1 0
0 0 1 1 0 1 1 0 1


Arithmetic Shift Right
MSB
LSB
7 6 5 4 3 2 1 0
1 1 0 1 1 0 1 0
1 1 1 0 1 1 0 1

Bitrotaties





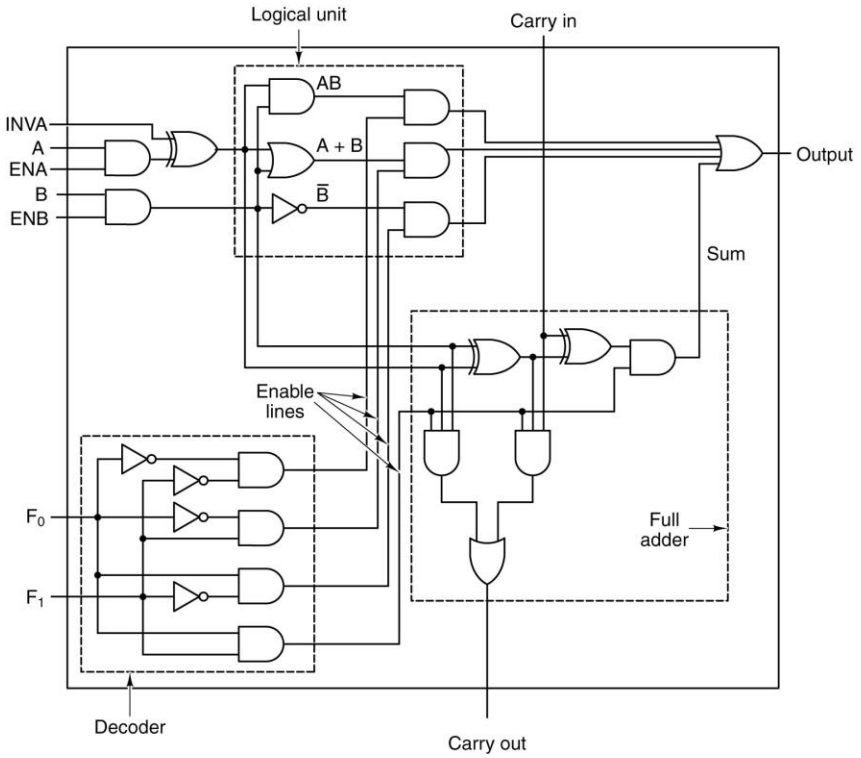
Adders

- basisbouwsteen voor optelling
- **half adder:** berekent som en carry-out van twee invoerbits
- **full adder:** berekent som en carry-out van twee invoerbits + carry-in
  - opgebouwd uit twee half adders

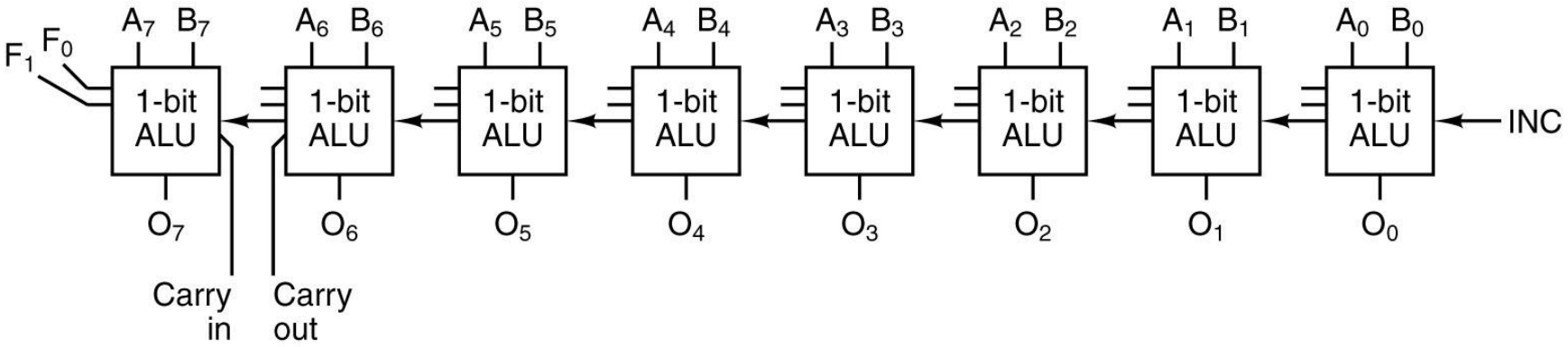
- practicum-processor:
  gebruik de kant en klare 32 bits adder

een ALU voor één bit

INVA	ENA	ENB	F0	F1	uitvoer
0	1	1	1	1	*B* + *A*
1	1	1	1	1	*B* – *A*
0	1	1	0	0	*A* AND *B*
0	1	1	0	1	*A* OR *B*
1	1	0	0	1	NOT *A*
x	x	1	1	0	NOT *B*
0	0	0	0	1	0
1	0	0	0	1	1



een ALU voor 8 bits



- complete ALU opgebouwd uit meerdere 1-bit ALUs
- INC = carry in voor laagste bit
  kan gebruikt worden voor increment (A+1, B+1)
- vermenigvuldiging, deling en bitshift in aparte eenheden

De ALU in de practicum-processor

- Hades bevat al 32-bit operaties (adder, bitwise or etc.)
- Maak direct een 32-bit ALU met deze componenten
- Lees de beschrijving van de practicumprocessor en de implementatie hints goed door.

De registers in detail

- opgebouwd uit flipflops

- vaak met meerdere aansluitingen
  - meerdere (interne) bussen
  - De ontwerper moet van tevoren bedenken hoeveel lees- en schrijfoperaties hij in dezelfde cyclus wil laten plaatsvinden
- in de practicum-processor te maken met:
  RegRE = register met reset- en enable-invoer
  (Uit rtlib.register)

de vlaggen in detail

- ALU produceert ook vlaggen
  - zero — $\overline{Out_0 + Out_1 + ... + Out_{31}}$
  - carry — Carry Out_{31} = Carry In_{32}
  - sign/negative — Out_{31}
  - overflow — Carry Out_{30} XOR Carry Out_{31} (Maar het kan slimmer)
- elke vlag = 1-bit flipflop
  - soms: alle vlaggen samen in een speciaal register

De besturingseenheid in detail

- Hoe wordt een programma uitgevoerd:
  1. instructie lezen (adres: programmateller) (fetch)
  2. programmateller ophogen (kan vaak tegelijkertijd)
  3. instructie decoderen (decode)
  4. instructie uitvoeren (execute)

RISC versus CISC

- Een CISC (Complex Instruction Set Computing) processor kan met 1 instructie diverse operaties of multi-step operaties uitvoeren
  - Veel oude processoren hadden zo'n architectuur (IBM 360, PDP-11, VAX, 80x86, ...)

- Later kwam het inzicht dat een simpele instructieset waarbij elke instructie slechts 1 operatie uitvoert, een hogere performance gaf.
  - Begonnen met de MIPS en SPARC begin 80er jaren (DEC Alpha, ARM, PA-RISC, Power,...)

Hardwired vs. Microprogrammed

Processoren komen in 2 soorten:

- Bij hardwired processoren wordt door middel van een (complexe) eindige automaat de instructies opgehaald, gedecodeerd en uitgevoerd.

- Veel Risc processoren zoals ARM, MIPS (en de practicum processor) werken op deze manier.

- Des te regelmatiger de ISA (Zie later), des te makkelijker een hardwired processor te maken is. We gaan dit op het werkcollege/practicum doen!

Hardwired vs. Microprogrammed 2

- Bij microgeprogrammeerde processoren is er een interne processor die een microprogramma uitvoert om de instructies te fetchen, te decoderen en uit te voeren.

- Dit interne processor interpreteert dus de inkomende instructies middels het microcode programma.

- Vrijwel alle CISC processoren zoals Intel zijn micro-geprogrammeerd.

De besturingseenheid in detail 2

- besturingseenheid kiest operatie
- afhankelijk van de mogelijke operaties moeten bepaalde verbindingen bestaan
  - interne bus: gegevenstransport binnen CPU
  - besturings-signalen / control signals:
    kies welke gegevens gelezen worden,
    kies ALU-operatie etc.,
    kies waar het resultaat wordt opgeslagen

Hoe kiest de besturingseenheid?

- schakelingen om keuze te ondersteunen
  - multiplexer:
    kies één van de ingangen,
    afhankelijk van besturingslijn
  - Tri-state buffer: schakel uitgang in of uit.
    De bedoeling is dat van meerdere tri-state buffers telkens maximaal één ingeschakeld is.
- Enable-invoer van registers etc.

Acties van de besturingseenheid: Fetch

- Zet de programmateller (PC) op de adresbus, start een leesoperatie op het geheugen.
- Reken met de ALU PC + 4 uit en schrijf dit terug in de PC
- Schrijf de instructie die uit het geheugen gelezen wordt in het instructieregister.

Acties van de besturingseenheid: Execute

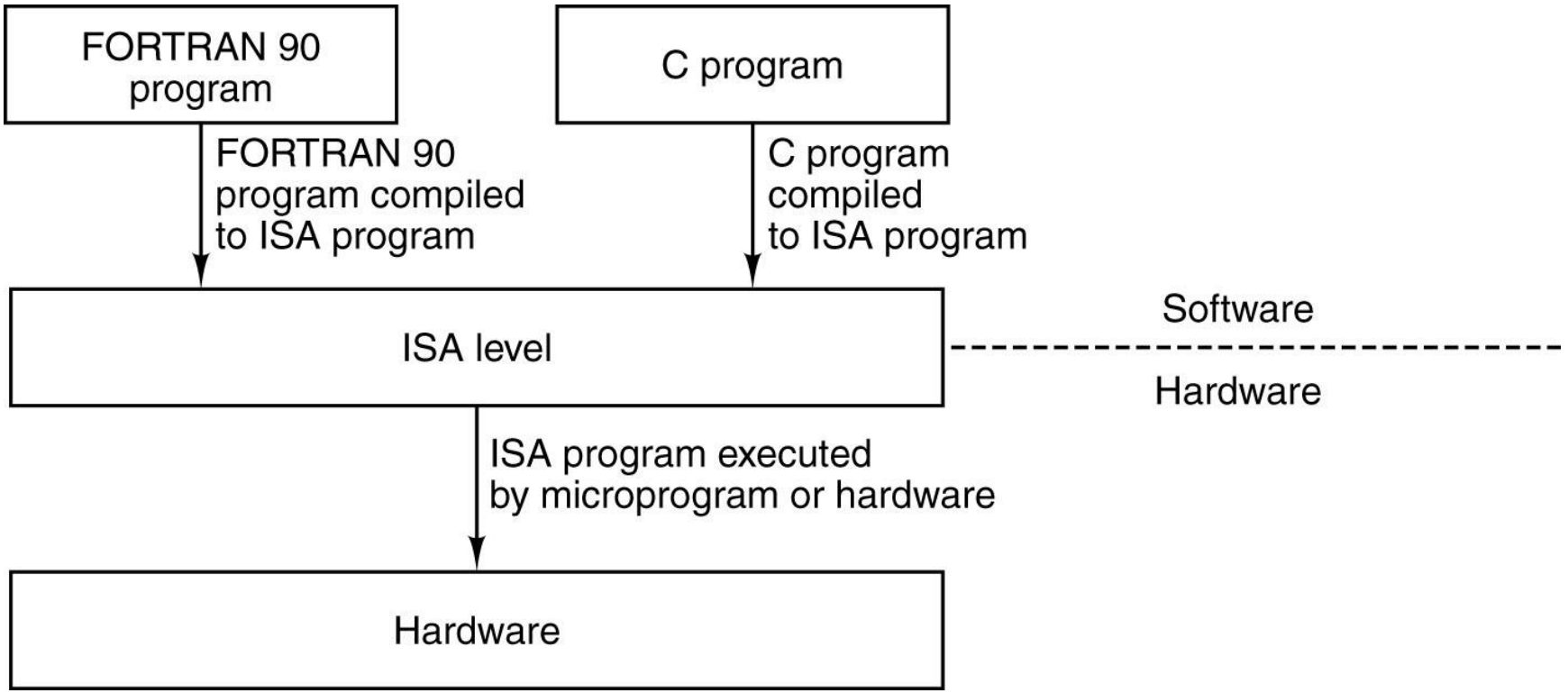
Bepaal het instructieformaat van de instructie in het instructieregister:

- Bepaal of deze instructie uitgevoerd moet worden
- Bepaal welke registers gelezen en welk register geschreven moet worden
- Bepaal hoe de interne bussen hun waarden moeten krijgen
- Bepaal welke operatie de ALU moet uitvoeren
- Bepaal welke geheugenoperatie er uitgevoerd moet worden (of helemaal niet).

Instruction Set Architecture

- interface tussen software en hardware
- keuze van opcodes
  = welke machinecode is welke operatie?
- instructieset en CPU opbouw hangen samen
- RISC versus CISC

ISA: tussen software en hardware



Keuzes in de ISA

- eenvoudige opbouw van besturingseenheid
- eenvoudige compiler
- compatibiliteit met oudere ISA

Voorbeelden van architecturen

- Accumulator-architectuur: b.v. 6502
  - alle rekeninstructies gebruiken de accumulator
- architectuur met gelijkwaardige registers: bv. 68000, ARM, MIPS
  - elk register kan bron of doel zijn
- gemengde architectuur: b.v. 8086
  - veel instructies met elk register
  - sommige instructies met vast register
    b.v. ROR register,CL CL = count register

Addressing Modes

- Waar mogen bron- en doeloperanden staan?
  - registers
  - constante
  - geheugen
    - vast adres
    - berekend adres, b.v. register + offset
  - accumulator
  - indirect adres (adres staat in het geheugen)
- Addressing mode = **manier** van operand

Addressing modes van de Practicum-processor

- constante (meestal $-2^9 \ldots +2^9-1$)
- register
- alleen READ en WRITE: geheugen
  - berekend adres: register+constante
  - vast adres: R0+constante

Voorbeeld: 8086-ISA

- grondslag van x86-architectuur

- voorbeeld ter illustratie
  (details van 8086 zijn geen tentamenstof
  – jullie moeten het principe kennen)

delen van 8086-ISA

bit 7 6 5 4 3 2 1 0

- formaat 1: 00 opc yy s
  - opc = operatie

opc	000	001	010	011	100	101	110	111
operatie	ADD	OR	ADC	SBB	AND	SUB	XOR	CMP

  - yy = addressing mode = soort operand

yy	00	01	10	11
doel	geheugen	register	AL / AX	andere operaties
bron	register	geheugen	constante	

    bij 00 en 01 volgt een mode byte met details
  - s = grootte: 0 = byte (8 bit), 1 = word (16 bit)

http://en.wikibooks.org/wiki/X86_Assembly/Machine_Language_Conversion

delen van 8086-ISA: mode byte

- mm rrr ggg
  - mm = welke soort geheugenoperand?
  - ggg = welke registers geven het adres aan?

alleen 16-bit-offset

mm ggg	000	001	010	011	100	101	110	111
00: offset=0	[BX+SI]	[BX+DI]	[BP+SI]	[BP+DI]	[SI]	[DI]	direct	[BX]
01: 8 bit ofs	o[BX+SI]	o[BX+DI]	o[BP+SI]	o[BP+DI]	o[SI]	o[DI]	o[BP]	o[BX]
10: 16 bit ofs	O[BX+SI]	O[BX+DI]	O[BP+SI]	O[BP+DI]	O[SI]	O[DI]	O[BP]	O[BX]
11: register	AX	CX	DX	BX	SP	BP	SI	DI

  - rrr = registeroperand

rrr	000	001	010	011	100	101	110	111
16-bit-register	AX	CX	DX	BX	SP	BP	SI	DI
8-bit-register	AL	CL	DL	BL	AH	CH	DH	BH

delen van 8086-ISA

- formaat 1a: 10001 yy s
  - MOV (gegevenstransport)
- formaat 1b: 1000000 s mm opc ggg
  - ADD etc. met een constante als bron

delen van 8086-ISA

- formaat 2: 010 op rrr
  - op = operatie

op	00	01	10	11
operatie	INC	DEC	PUSH	POP

  - rrr = operand (altijd een register)

rrr	000	001	010	011	100	101	110	111
register	AX	CX	DX	BX	SP	BP	SI	DI

delen van 8086-ISA

- formaat 2a: 1011 s rrr
  - MOV register, constante
- formaat 2b: 10010 rrr
  - XCHG register, AX

small is beautiful

- slimme combinatie van operaties
  → kleinere ALU
- systematische ISA
  → kleinere besturingseenheid
  → eenvoudigere compiler