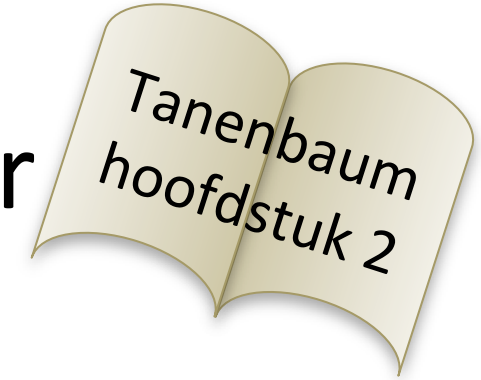
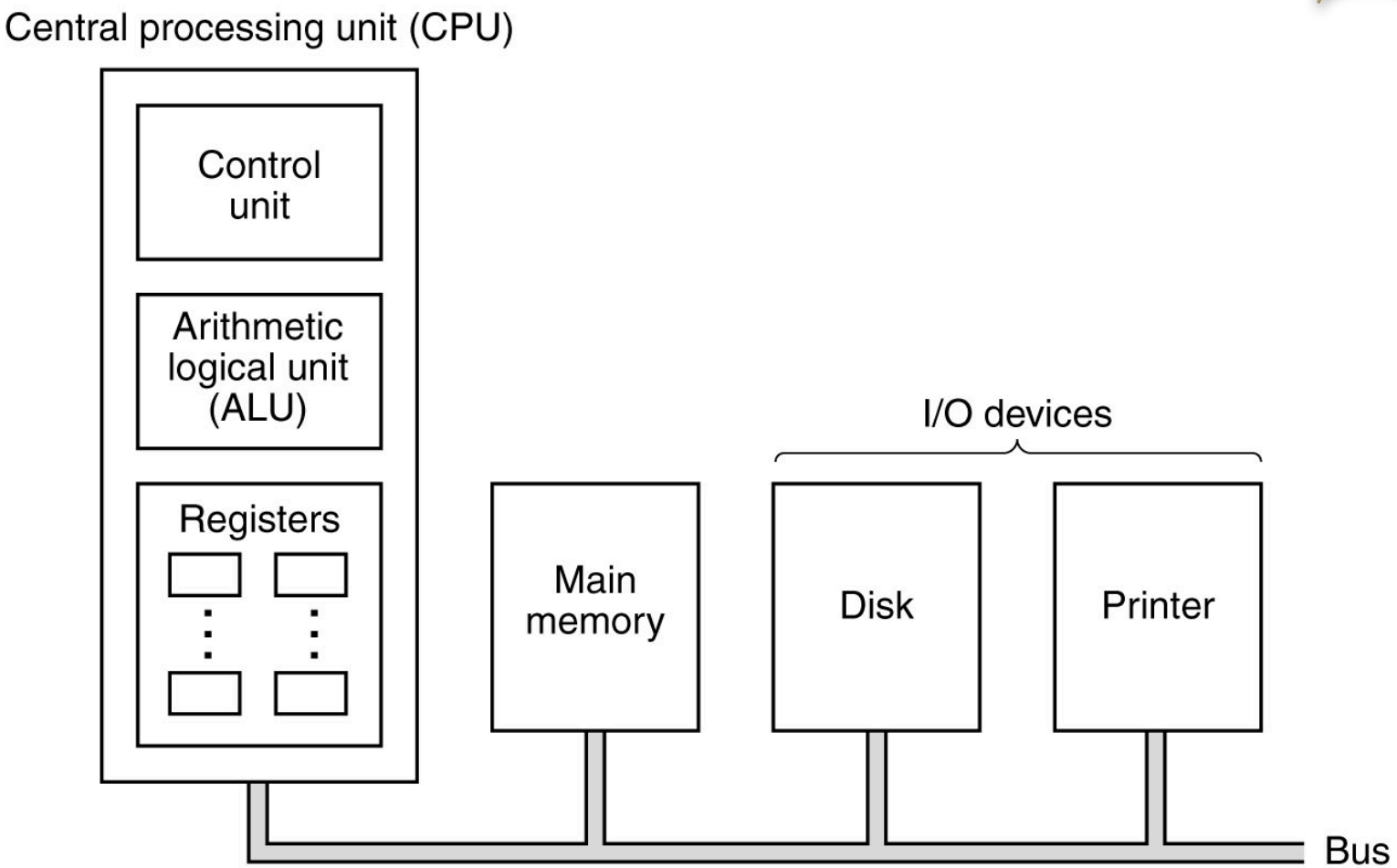
De CPU in detail
Hoe worden instructies uitgevoerd?

Processoren

28 februari 2012

„von Neumann”-architectuur



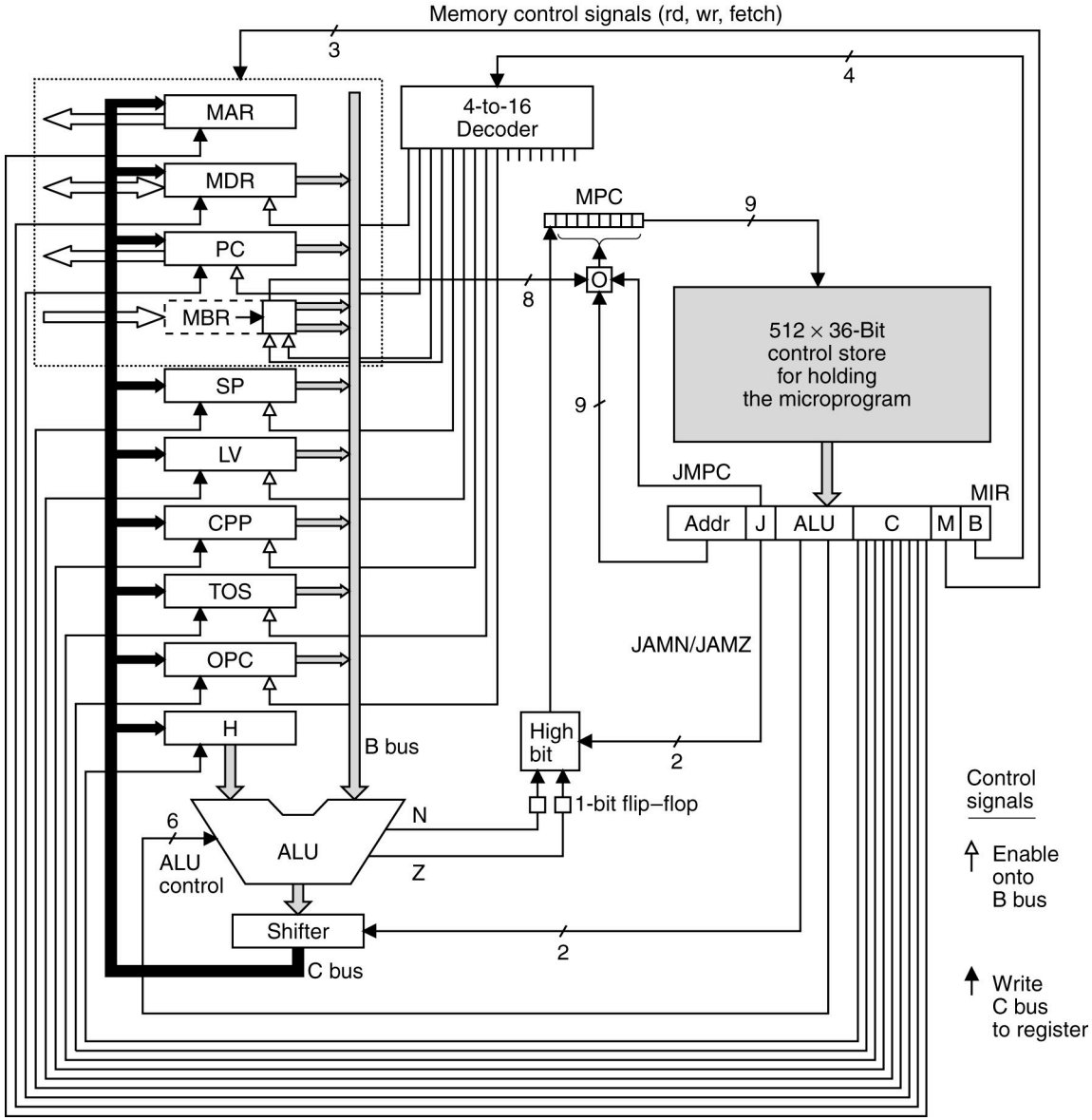


Structuur van de CPU

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

Microinstruction Control: The Mic-1 (1)

The complete block diagram of our example microarchitecture, the Mic-1.



de ALU in detail

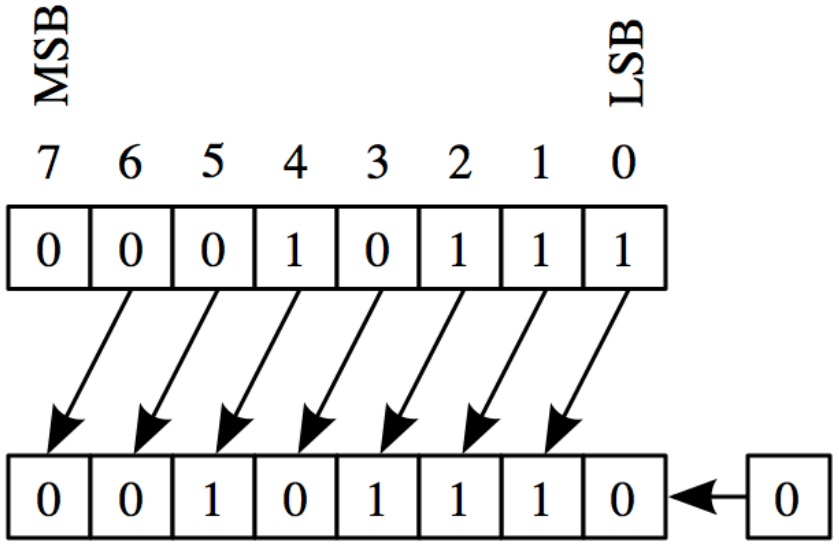
- Welke operaties moet de ALU aankunnen?
  - rekenoperaties: +, –, ×, ÷
  - logische operaties: AND, OR, NOT, XOR
  - bitshifts en bitrotaties

- efficiënte implementatie van subtractie:

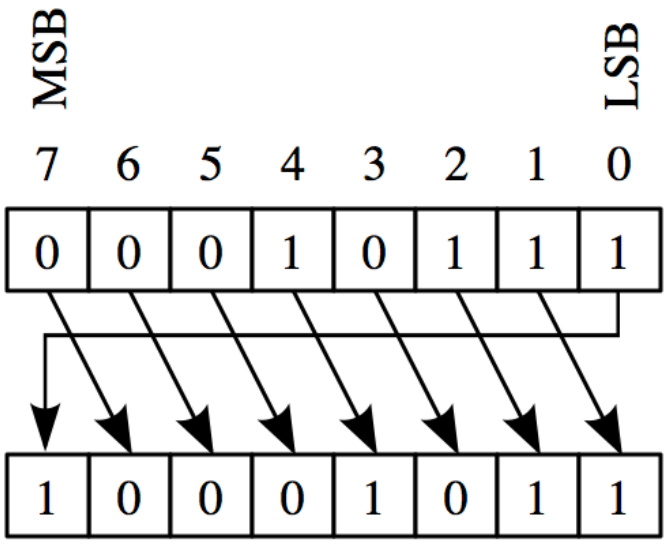
  $A - B = A + (-B) = A + (\text{NOT } B) + 1$

Bitshift en bitrotatie

Shift left



Rotate right



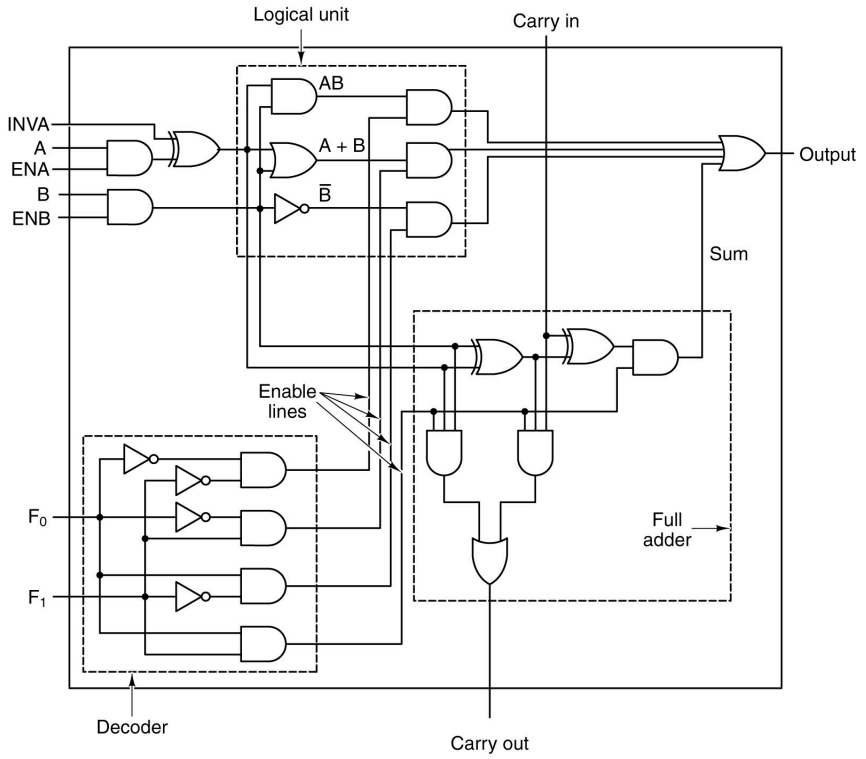
Addeerwerk

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

- practicum-processor: gebruik het kant-en-klare addeerwerk

een ALU voor één bit

INVA	ENA	ENB	F0	F1	uitvoer
0	1	1	1	1	*A* + *B*
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*
?	?	1	1	0	NOT *B*
0	0	0	0	1	0
1	0	0	0	1	1



De ALU in de practicum-processor

- Colibri bevat al 32-bit-operaties (addeerwerk, bitwise or etc.)
- maak direct een 32-bit-ALU!

de registers in detail

- opgebouwd uit flipflops,
  als een klein stukje RAM
- vaak met meerdere aansluitingen
  - meerdere (interne) bussen

- in de practicum-processor:
  RegRE = register met reset- en enable-invoer

de vlaggen in detail

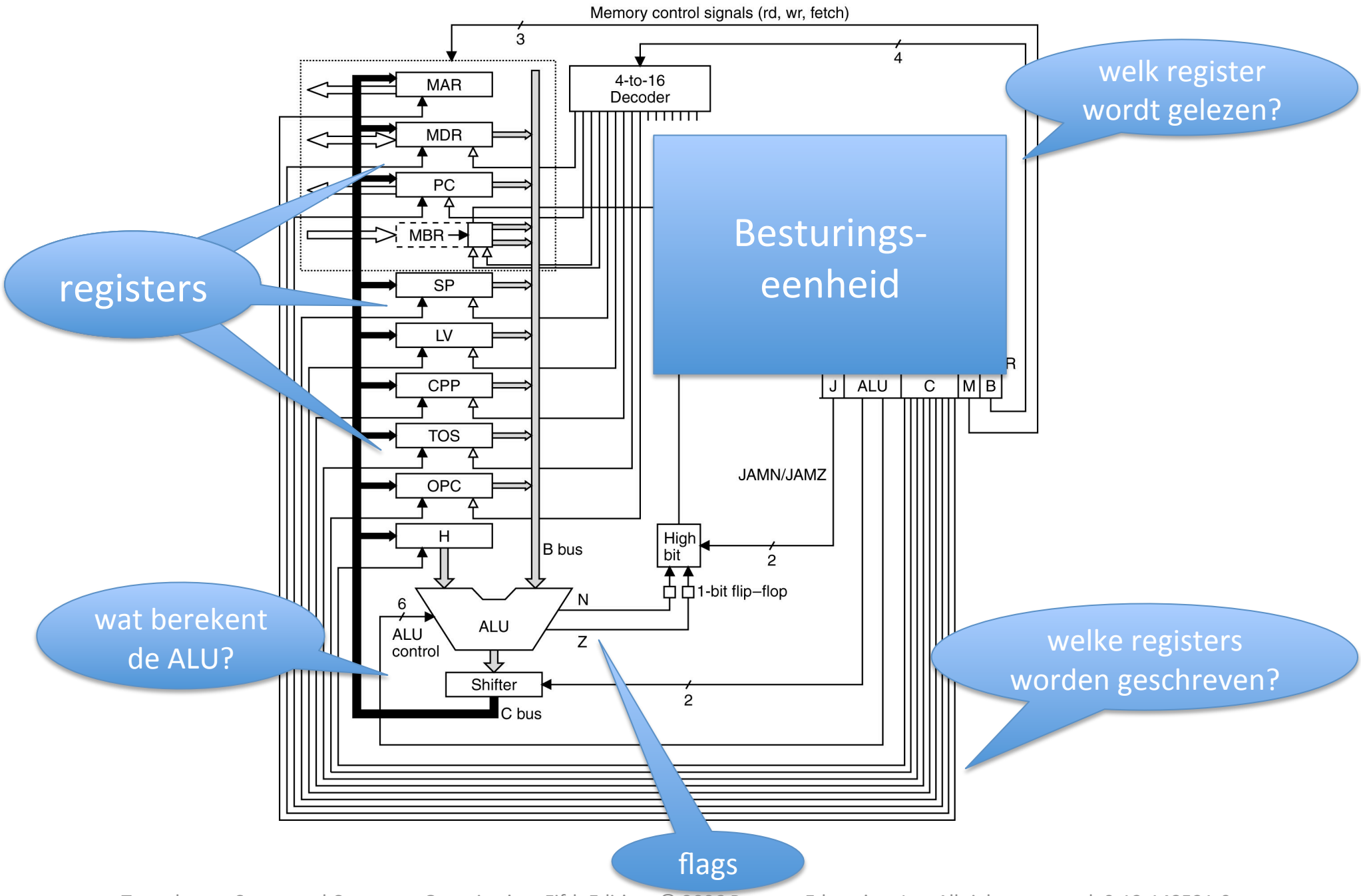
- ALU produceert ook vlaggen
  - zero $\neg(Out_0 \vee Out_1 \vee \ldots \vee Out_{31})$
  - carry $Carry_{31}$
  - sign/negative Out_{31}
  - overflow $Carry_{30}$ XOR $Carry_{31}$
- elke vlag = 1-bit-flipflop
  - soms: alle vlaggen samen in een speciaal register

de besturingseenheid in detail

- 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?

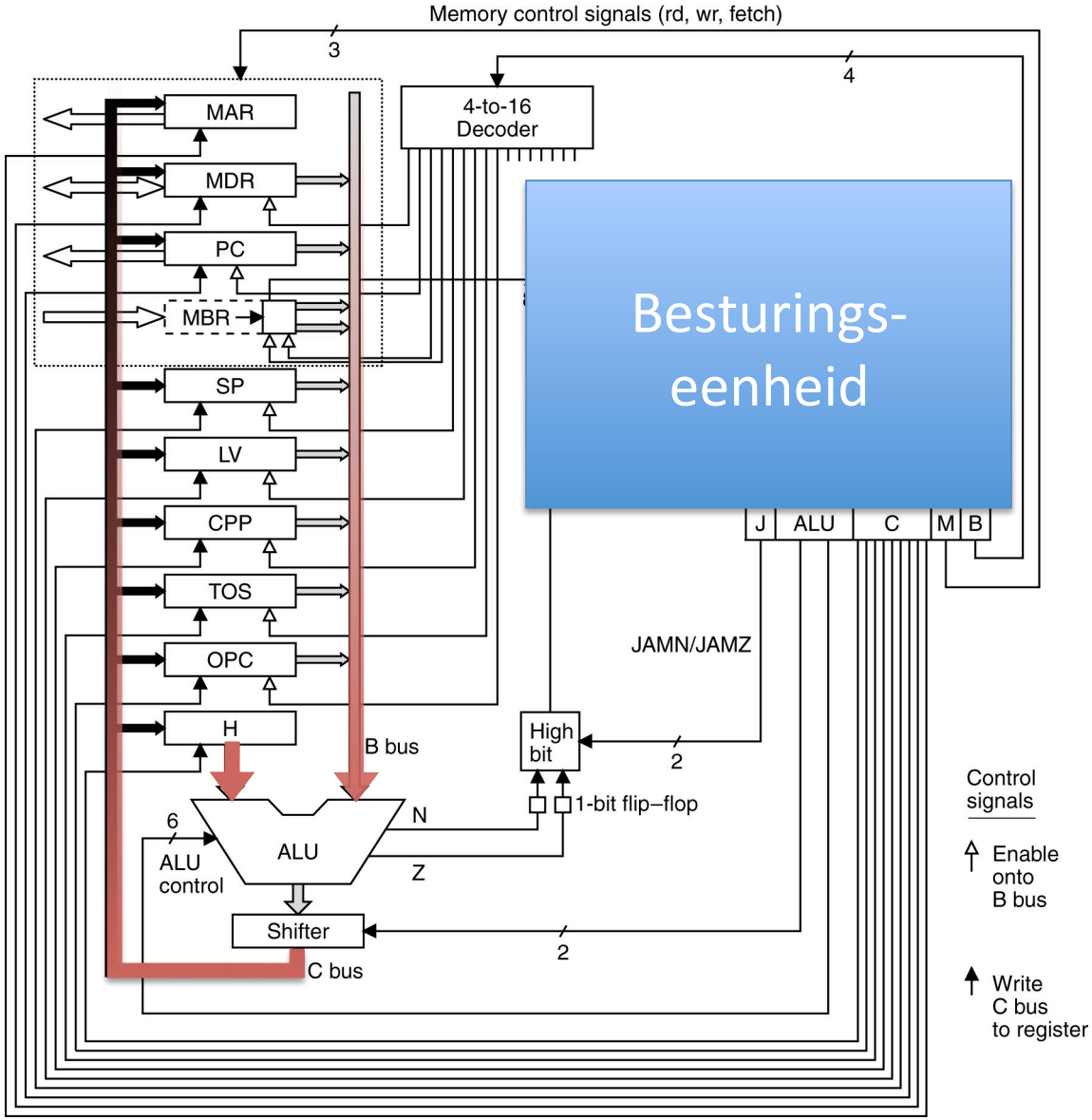
- schakelingen om keuze te ondersteunen
  - multiplexer:
    kies één van de ingangen,
    afhankelijk van besturingslijn
  - Tri-state buffer: schakel uitgang in of uit.
    De bedoeling is dat van meerdere tri-state buffers telkens maximaal één ingeschakeld is.
- Enable-invoer van registers etc.



Rekenoperatie

Datastromen:
van registers naar ALU
van ALU naar registers

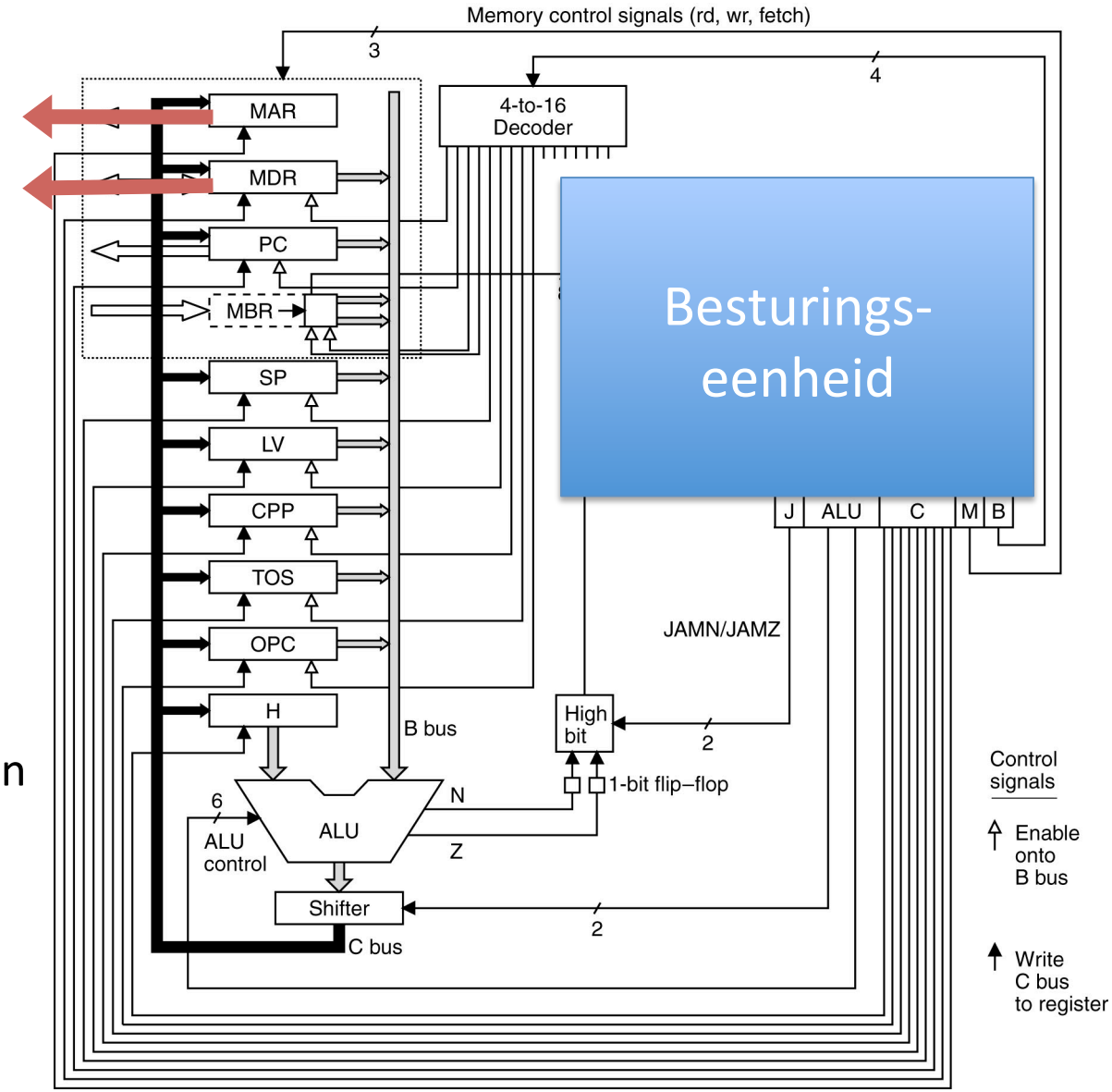
Besturing:
kies welke registers gelezen
en geschreven worden
kies ALU-operatie



In RAM schrijven

Datastromen:
van register naar
adres- en databus

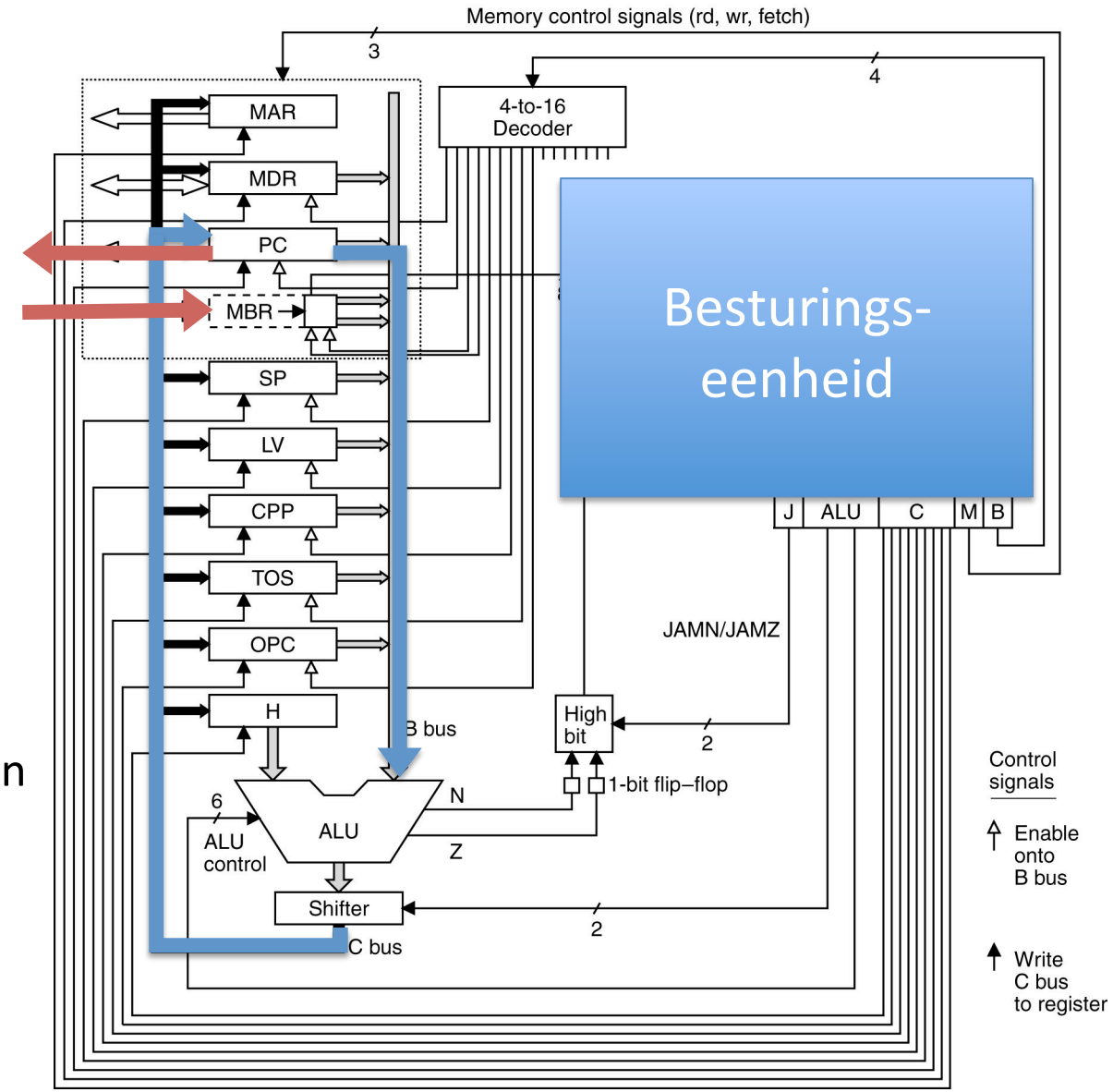
Besturing:
kies welke registers gelezen worden
RAM-operatie: schrijven



Instruction fetch + PC increment

Datastromen:
van register naar adresbus
van databus naar register

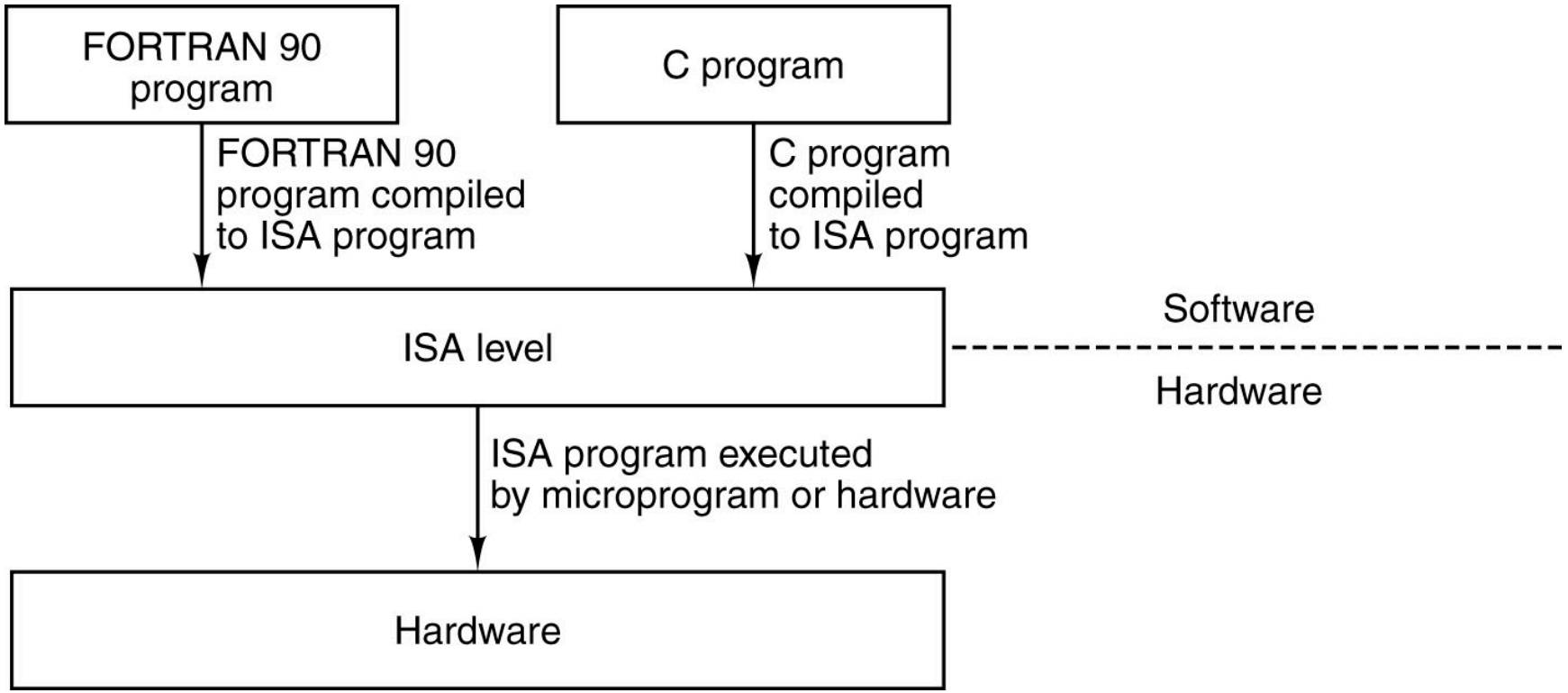
Besturing:
kies welke registers gelezen of geschreven worden
RAM-operatie: lezen
ALU-operatie: B+1



Instruction Set Architecture

- interface tussen software en hardware
- keuze van opcodes
  = welke machinecode is welke operatie?

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: b.v. 68000
  - 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

in oude processoren
met kleine registers

Addressing modes van de Practicum-processor

Addressing modes van de Practicum-processor

- constante (meestal $-2^9 ... +2^9-1$)
- 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	

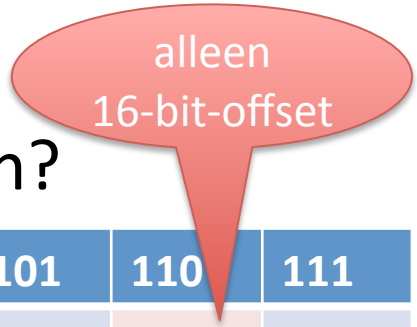
    bij 00 en 01 volgt een mode byte met details

  - 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?



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