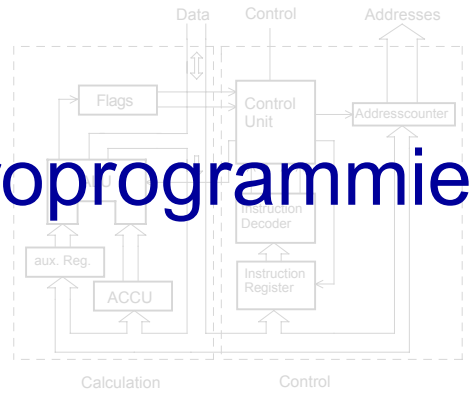
# Datenverarbeitung I

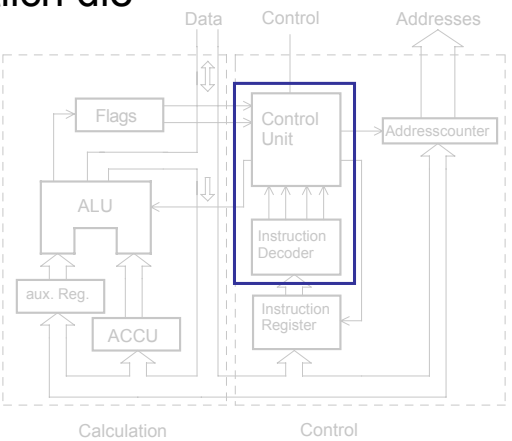



## Mikroprogrammierung



# Mikroprogrammierung

Wie werden eigentlich die Maschinenbefehle ausgeführt ?




Es gibt (mindestens) zwei Möglichkeiten.
Hier jedoch von Interesse:
Mikroprogrammierung.

# Mikroprogrammierung

Zuerst: Vorstellung eines einfachen Computers

Hypothetischer Minicomputer

nach

Andrew S. Tanenbaum

*Structured Computer Organization*

1st Edition 1976, Prentice Hall

Der Minicomputer ist die Zielmaschine (target machine), die es durch eine Wirtsmaschine (host machine) und entsprechender Mikroprogrammierung zu implementieren gilt.



# Mikroprogrammierung

Begriffe:

Target Level

Hardwarefunktionalität auf Maschinensprachebene. Blick auf Prozessorfunktionen aus Sicht des (Assembler)programmierers (= Programmiermodell).

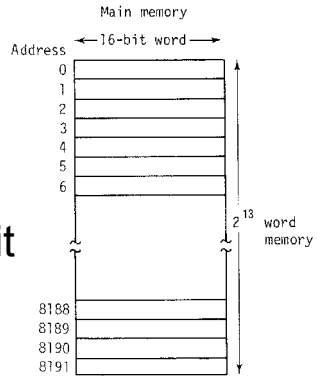
Host Level

Hardwarefunktionalität auf Register-Transfer Ebene. Blick auf den Prozessor aus Sicht des Prozessor-entwicklers / Mikroprogrammierers.

# Target Machine



- Wortbreite 16 Bit
  Keine `bytes`, nur `words`.
- Adressbusbreite 13 Bit
- Speicher: $2^{13}$ Worte à 16 Bit
  Daten nur `word`-weise ansprechbar.
- Stackmaschine
  Arithmetische Operationen werden auf dem Stack ausgeführt.
  - Operanden vom Stack holen
  - Arithmetische Operation durchführen
  - Ergebnis auf Stack legen

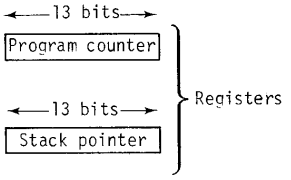





# Target Machine



- Nur zwei Register auf Maschinenebene
  - Program counter (`PC`)
  - Stack pointer (`SP`)
- Stack arbeitet ‚reverse'
  Stack wächst in Richtung aufsteigender Adressen!
- Befehlssatz: 12 Maschinenbefehle
  Notation für nachfolgend dargestellten Befehlssatz:
  `M` = Speicheradresse (13 Bit)
  `T` = oberstes Wort auf dem Stack (16 Bit)
  `S` = zweitoberstes Wort auf dem Stack (16 Bit)



# Befehlssatz



- Stackbefehle (Anzahl 2)

  Speicherdirekte Adressierung, 1-Adress-Befehle, Adresse = M.

  - PUSH M

    Der Inhalt der Speicherzelle M wird auf den Stack gelegt.

  - POP M

    Das oberste Wort wird vom Stack in Speicherzelle M verschoben.

- Unterprogrammbefehle (Anzahl 2)

  - CALL M

    Springe zu M (PC := M), zuvor wird Rücksprungadresse auf Stack gesichert. 1-Adress-Befehl. Speicherdirekte Adressierung.

  - RETURN

    Hole Wort T vom Stack (Rücksprungadresse). Springe zu T (PC:=T).



# Befehlssatz



- Sprungbefehle (Anzahl 4)

  Speicherdirekte Adressierung, 1-Adress-Befehle, Adresse = M.

  - JUMP M

    Springe zu M (PC:=M).

  - JNEG M

    Hole oberstes Wort T vom Stack. Wenn T < 0 dann springe zu M.

  - JZER M

    Hole oberstes Wort T vom Stack. Wenn T = 0 dann springe zu M.

  - JPOS M

    Hole oberstes Wort T vom Stack. Wenn T ≥ 0 dann springe zu M.

# Befehlssatz



- Arithmetische Befehle (Anzahl 4)

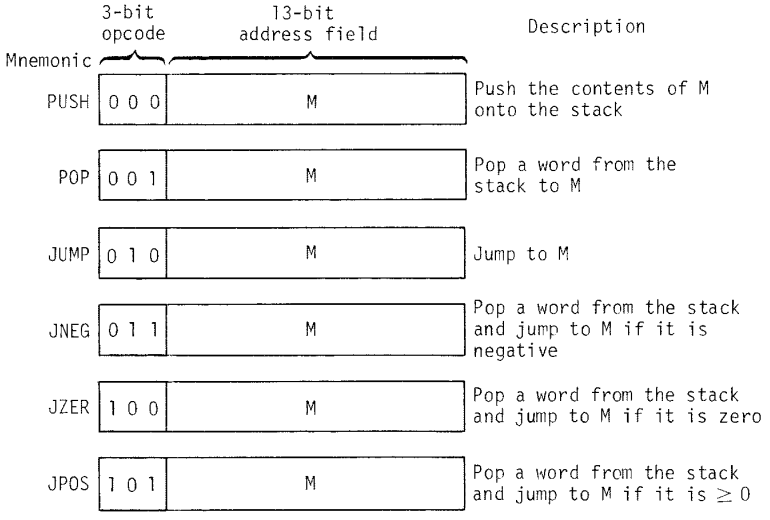
  0-Adress-Befehle.

  - `ADD` (Addition)
    Pop `T`, Pop `S`, Push (`S+T`)
  - `SUB` (Subtraktion)
    Pop `T`, Pop `S`, Push (`S-T`)
  - `MUL` (Multiplikation)
    Pop `T`, Pop `S`, Push (`S*T`)
  - `DIV` (Division)
    Pop `T`, Pop `S`, Push (`S/T`)



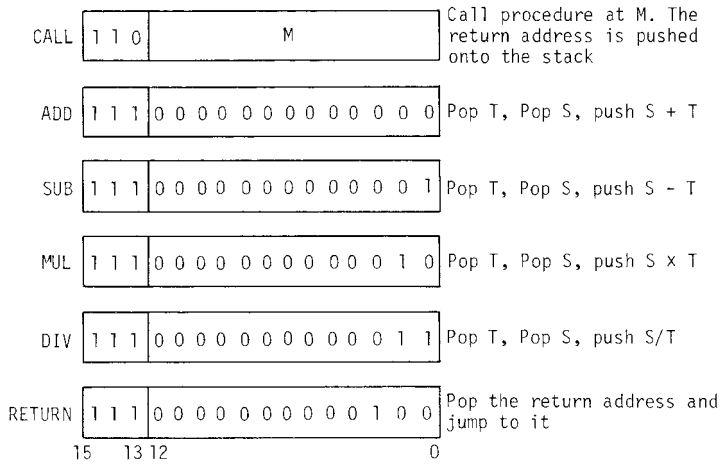
Befehlssatz

Target Machine



# Befehlssatz



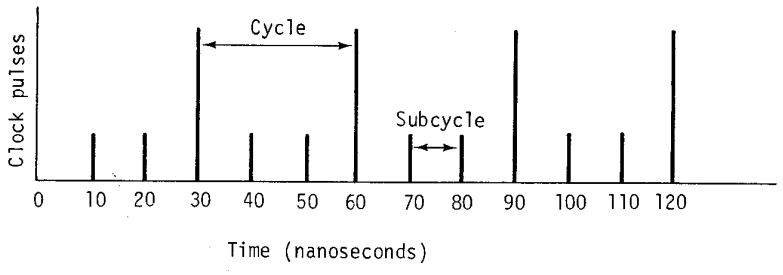





# Target Machine



- Taktzyklus 30 ns (target machine cycle)
- Taktzyklus intern unterteilt in drei Unterzyklen zu je 10 ns.



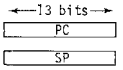
# Mikroprogrammierung

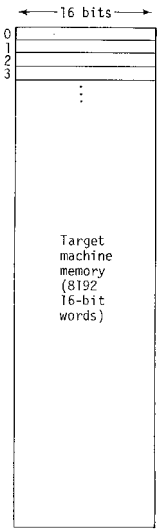
Bis jetzt:

Minicomputer vorgestellt
(Programmiermodell, Target Level)

Nun:

- Implementierung des Prozessors
- „Zoomen“ auf Host Level
  Das Register-Transfer-Schema wird sichtbar.






# Host Machine



Neue Hardwarekomponenten sichtbar:

- Hilfsregister
  - Pufferregister für Datenbus und Adressbus
    `MBR` (Memory Buffer Register), `MAR` (Memory Address Register)
  - Pufferregister für Maschinenbefehl
    `IR` (Instruction Register)
  - Diverse Zwischenspeicher
    Register `A`, `B`, `C`, `D`
  - Zählregister für Mikroprogrammschleifen
    Register `X`

# Host Machine



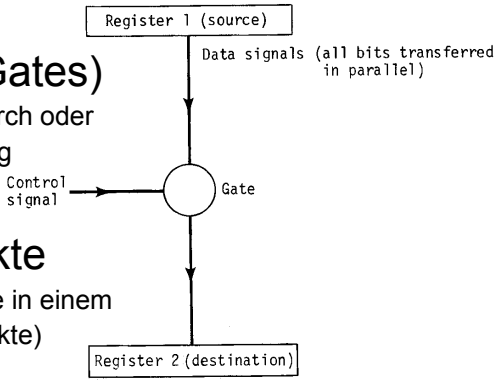
- Festwertregister
  Fünf Register mit konstanten Werten
  - Konstanten: `0`, `-1`, `+1`, `15`, `-32768` (`SIGN`)
  - Register nur lesbar (Read-only)
- 16-Bit Addierer
- 1-Bit Shifter
- Einerkomplement Bildung



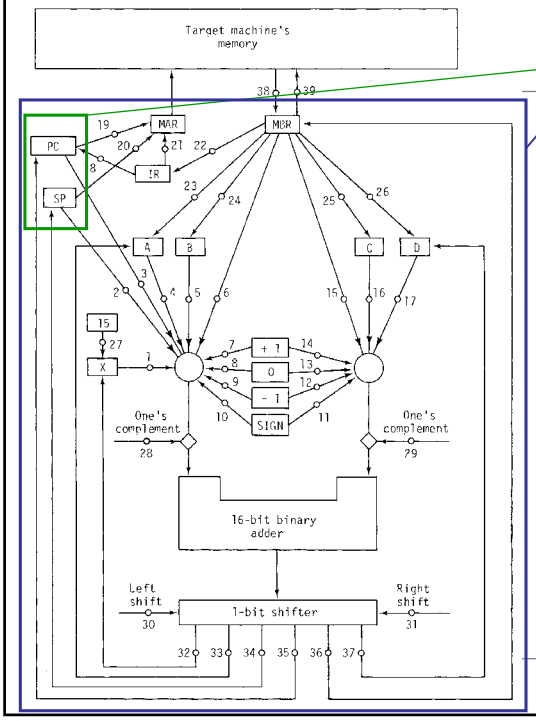
# Host Machine

Mikroprogrammierung

- Verbindungsbusse
- Transitschalter (Gates)
  Schalten Busverbindung durch oder unterbrechen Busverbindung
- Vereinigungspunkte
  Führen ankommende Busse in einem Bus weiter (Konvergenzpunkte)




Vorlesung Datenverarbeitung 1 WS 2005/2006 Dr.-Ing. Stefan Freinatis




Registermodell (Target Level)

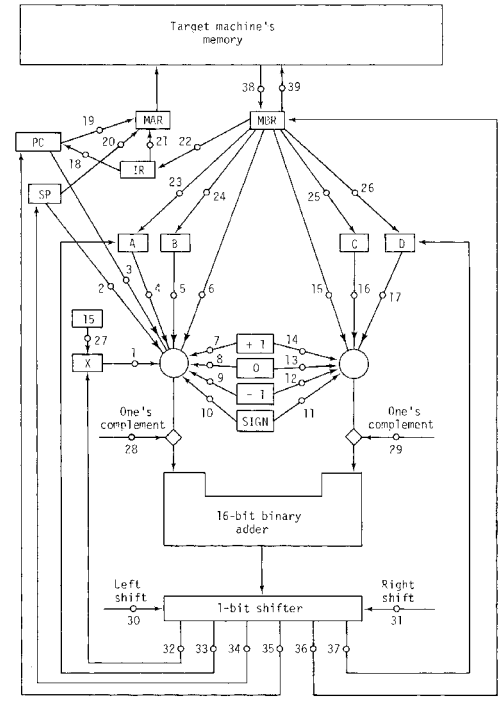
Registermodell (Host Level)

- 45 Busverbindungen
- 39 Transitschalter (Gates)
- 2 Vereinigungspunkte

Kontrollsignalflüsse nicht dargestellt, nur Datenflüsse.

Implementierung der Maschinenbefehle durch gezieltes Öffnen und Schließen der Gates.






# Signallaufzeit

1. Unterzyklus

Register-zu-Register Transfers

Nur Gates 1 bis 29 dürfen benutzt werden.

2. Unterzyklus

Ergebnisse Addition/Shiften

Nur Gates 30 bis 37 dürfen benutzt werden.

3. Unterzyklus

Lesen/Schreiben Speicher

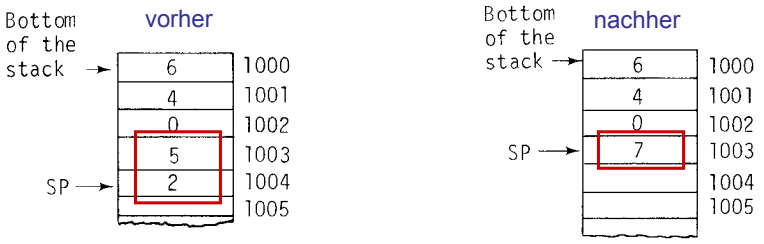
Nur Gate 38 oder Gate 39 dürfen benutzt werden.

Implementierung der Maschinenbefehle durch gezieltes Öffnen und Schließen der Gates. ⇒

## Beispiel: ADD Befehl

ADD = Pop T, Pop S, Push (S+T)

Der oberste Wert wird vom Stack geholt. Der Stackpointer SP wird dekrementiert und der zweite Wert vom Stack geholt. Beide Werte werden addiert und das Ergebnis wieder auf dem Stack abgelegt. Nach Ausführung des Maschinenbefehls besitzt der Stack ein Element weniger als vorher (2 x POP, 1 x PUSH).





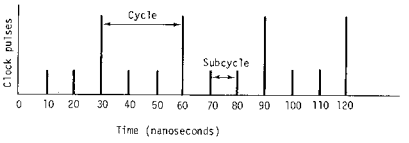



## Implementierung ADD Befehl

durch Steuerung der Gates (Datenflusssteuerung auf Host Level)

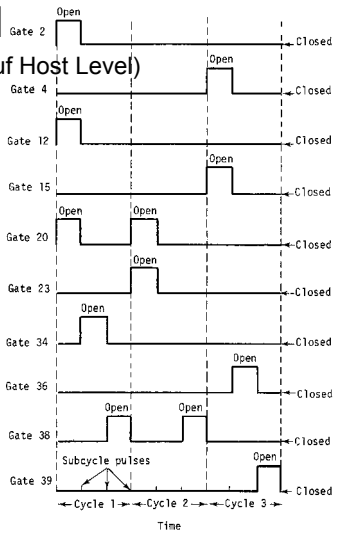
Annahmen für beispielhafte Implementierung:

- Befehlscode ADD (E000 H) ist schon geholt und dekodiert.
- Zugriffszeit auf Speicher: 1 Unterzyklus.
- Instruction Fetch separat vom ADD-Befehl
- Alle Gates sind geschlossen



# Implementierung ADD Befehl

durch Steuerung der Gates (Datenflusssteuerung auf Host Level)




Taktzyklus 1

- Gate 20
  SP → MAR (SP auf Adressbus)
- Gate 38
  Start Memory Read. Top of Stack (T) geht in Zyklus 2.1 in MBR.
- Gates 2, 12, 34
  Dekrementieren von SP: SP in Addierer (rechts), Konstante −1 in Addierer (links), Ergebnis über Gate 34 zurück in SP.



ADD

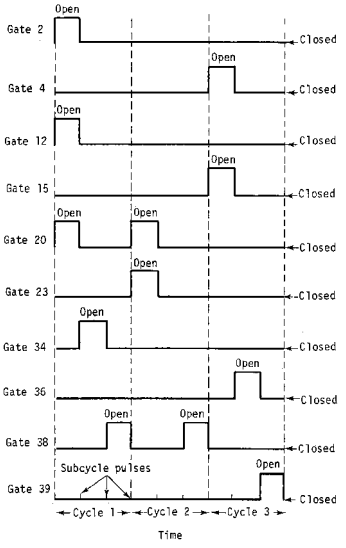
Host Machine

Taktzyklus 2

- Gate 23
  MBR → A (Zwischenspeichern T in A)
- Gates 20, 38
  SP → MAR (SP auf Adressbus). Start Memory Read. S → MBR (Zyklus 3.1).

Taktzyklus 3

- Gates 4, 15
  T und S gehen in Addierer.
- Gates 36, 39
  Summe (S+T) über Gate 36 ins MBR, dann über Gate 39 in Speicher (Stack).




Vorlesung Datenverarbeitung 1 WS 2005/2006 Dr.-Ing. Stefan Freinatis

# ADD



| | Subcycle | SP | MAR | MBR | A |
|---|---|---|---|---|---|
| Cycle 1 | 1 | 1004 | ? | ? | ? |
| | 2 | 1004 | 1004 | ? | ? |
| | 3 | 1003 | 1004 | ? | ? |
| Cycle 2 | 1 | 1003 | 1004 | 2 | ? |
| | 2 | 1003 | 1003 | 2 | 2 |
| | 3 | 1003 | 1003 | 2 | 2 |
| Cycle 3 | 1 | 1003 | 1003 | 5 | 2 |
| | 2 | 1003 | 1003 | 5 | 2 |
| | 3 | 1003 | 1003 | 7 | 2 |
| After cycle 3 | | 1003 | 1003 | 7 | 2 |

Register `SP`, `MAR`, `MBR` und `A` während Ausführung `ADD`

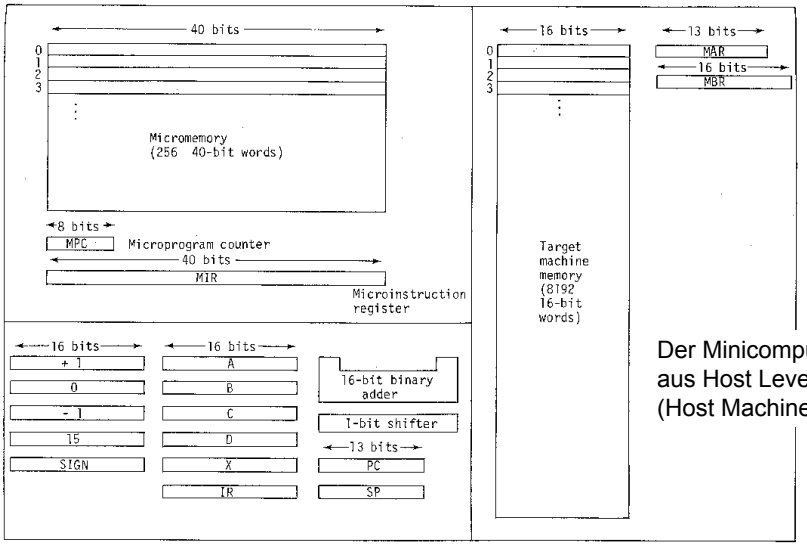


# Mikroprogrammierung

Steuerung der Gates mittels

- fest verdrahteter Hardware
  Zählerschaltungen mit Decodern (timing rings)
- Mikroprogrammierung
  Steuerung der Gatter durch Mikrobefehle (micro instructions)
  - Zwei Mikrobefehle: `GATE` und `TEST`
  - Mikroprogramm im Mikroprogramm-speicher (micro memory) in der CPU.

# Mikroprogrammierung

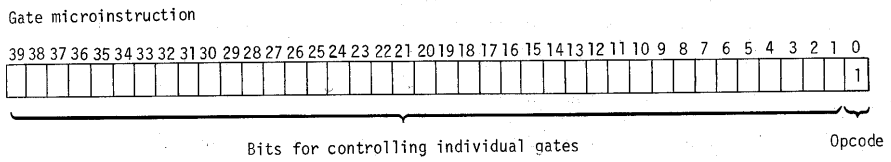



Der Minicomputer aus Host Level Sicht (Host Machine)



# Mikrobefehl GATE

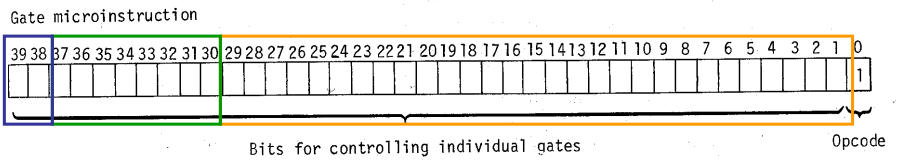





- Opcode = 1 (Bit 0)
- Steuert die 39 Gates (Bit 1 … 39)
  1 = „Gate on“ (Transitschalter geschlossen)
  0 = „Gate off“ (Transitschalter offen)
- Mehrere Gates gleichzeitig steuerbar
  Parallele Datenflusssteuerung auf Host Level

# Mikrobefehl GATE



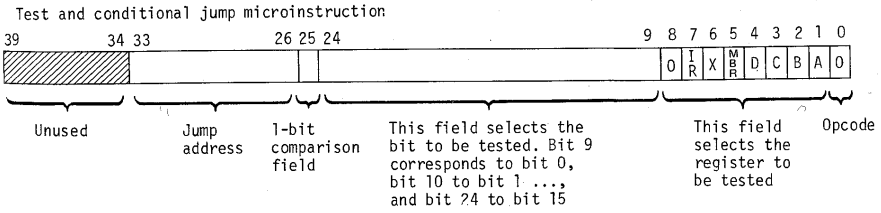



- Abarbeitung in einem Hauptzyklus
  aber nicht alle Gates gleichzeitig, sondern
  - Unterzyklus 1: Gates 1 bis 29
  - Unterzyklus 2: Gates 30 bis 37
  - Unterzyklus 3: Gate 38 oder 39



# Mikrobefehl TEST

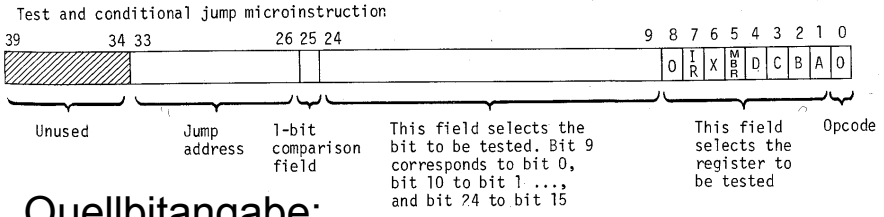





- Opcode = 0 (Bit 0)
- Testet ein wählbares Bit (Quellbit) auf Gleichheit mit Bit 25
- Bei Gleichheit: Sprung

# Mikrobefehl TEST






## Quellbitangabe:

- Bit 1 bis 8 spezifizieren Register
  Nur ein Bit darf gesetzt sein. Register `0` für unbedingte Sprünge.
- Bit 9 bis 24 spezifizieren eines der 16 Bit im gewählten Register
  Nur ein Bit in [9 … 24] darf gesetzt sein.



# Mikroprogrammierung

## Mikrobefehle = 40-Bit Worte

Nicht unbedingt leserlich für den Menschen

```
0100010000 0000000010 0000010000 0000000101
1000000000 0000000001 0000000000 0000000001
0000000000 0000000000 0000000000 1100000000
```

## Einführung „Mikroprogrammiersprache“

- Keine prozessorübergreifende Norm
  Wie auch bei Maschinensprache (Assembler)
- Mehrere „Statements“ pro Mikrobefehl möglich
  Nur für `GATE` Befehle. Verschiedene Gates gleichzeitig schaltbar.

# MPL



Andrew S. Tanenbaum:

„MPL“ (MicroProgramming Language)

- Notation angelehnt an Programmiersprache PL/1
- Beschreibung mittels Anweisungen
- Jede Textzeile beschreibt einen Mikrobefehl
- Mehrere Anweisungen pro Zeile möglich
  Nur bei `GATE` Befehlen.
- Reihenfolge Anweisungen innerhalb Zeile egal



# MPL (Beispiele)



## Transfer

- `A = MBR;` `/* Gate 23 */`
- `A = MBR; IR = MBR; X = 15;` `/* Gates 23,22,27 */`
- `A = MBR; IR = MBR; X = 15;` `/* Gates 23,22,27 */`
  `IR = MBR; X = 15; A = MBR;` `/* Gates 23,22,27 */`
- `MBR = Memory(MAR);` `/* Gate 38 */`
  Read from memory
- `Memory(MAR) = MBR;` `/* Gate 39 */`
  Write to memory

# MPL (Beispiele)



## Addition

- `MBR = A + C;` `/* Gates 4,16,36 */`
- `SP = SP + (-1)` `/* Gates 2,12,34 */`
- `D = 0 + C` `/* Gates 8,16,37 */`
- `A = MBR; PC = PC + 1;` `/* Gates 3,14,23,35 */`

## 1er Komplement

- `MBR = COM(MBR);` `/* Gates 6,13,28,36 */`
- `MBR = COM(MBR)+1;` `/* Gates 6,14,28,36 */`



# MPL (Beispiele)



## Shiften

- `MBR = LEFT_SHIFT(1+1);` `/* Gates 7,14,30,36 */`
- `A = RIGHT_SHIFT(A+0);` `/* Gates 4,13,31,33 */`

## Testen und Springen

- `IF BIT(13,IR) = 1 THEN GOTO POP;`
- `IF BIT(15,X) = 0 THEN GOTO MULLOOP;`
- `GOTO MAINLOOP` `/* unconditional jump*/`

# Mikroprogramm



Mikroprogramm = „Betriebssystem“ der CPU

Steuerung der Gates und Testen von Bits in Registern

Ausführung Maschinenprogramm:

- Holen Maschinenbefehl in `IR`
- Auswertung des Befehls (Testen bestimmter Bits in `IR`)
- Verzweigung zu entsprechender Routine. Ausführung.



# Mikroprogramm



```
ADDRESS         STATEMENT

    0     MAINLOOP:  MAR= PC; MBR= MEMORY (MAR); /* FETCH NEXT TARGET INSTR */
    1                IR= MBR; PC= PC + 1; /* MOVE INSTR TO IR AND ADVANCE PC */
    2                IF BIT(15,IR)= 1  THEN GO TO OP4567; /* DETERMINE TYPE */
    3                IF BIT(14,IR)= 1  THEN GO TO OP23;
    4                IF BIT(13,IR)= 1  THEN GO TO POP;

    5     PUSH:      MAR=IR; SP= SP + 1; MBR= MEMORY (MAR);
    6                MAR =SP; MEMORY (MAR) = MBR;
    7                GO TO MAINLOOP;

    8     POP:       MAR = SP;  SP = SP + (-1); MBR = MEMORY(MAR);
    9                MAR=IR; MEMORY(MAR) = MBR;  /* MAR=IR IS 13 BITS WIDE */
   10                GO TO MAINLOOP;

   11     OP23:      IF BIT(13,IR) = 1  THEN GO TO JNEG;

   12     JUMP:      PC = IR;      /* THIS DATA PATH IS 13 BITS WIDE */
   13                GO TO MAINLOOP;
   ...               ...
```

Vollständiges Mikroprogramm: siehe Hilfsblätter

# Mikroprogrammierung

Before 1951, the control logic for central processing units was designed by *ad hoc* methods. One of the simplest was to use rings of flip-flops to sequence the computer's control logic.

In 1951 Maurice Wilkes had a fundamental insight. He realized that if one takes the control signals for a computer, one could understand them as being played much like a player piano roll. That is, they are controlled by a sequence of very wide words constructed of bits.

Quelle: Wikipedia (http://en.wikipedia.org/wiki/microprogram)



# Vorteile



- **Relativ wenig Mikrobefehle nötig**
  Durch Schleifen und Sprünge können Mikrobefehle während der Ausführung eines Maschinenbefehls mehrfach genutzt werden.
- **Geringere Kosten**
  Weniger Hardware nötig, geringerer Platzbedarf
- **Überschaubarer**
  Für den Hardware-Entwickler überschaubarer als ein hart verdrahtetes Schaltwerk mit unzähligen Zählern, Dekodern und Verbindungsleitungen.
- **Maschinenbefehlssatz änderbar**
  Durch Änderung/Austausch des Mikroprogramms kann der Befehlssatz auf Target Level geändert oder ausgetauscht werden. Das geht natürlich ...

# Mikroprogrammierung

... nur bei mikroprogrammierbaren Prozessoren.
Mikroprogrammierbar: Mikroprogramm ist in einem EPROM. Kann vom Anwender geändert werden.
Mikroprogrammiert: Mikroprogramm ist im ROM. Änderungen nur durch Hersteller (Neuauflage).

## Nachteile Mikroprogrammierung

- Langsamer
- Verleitet zu komplexen Befehlen (CISC)
  CISC = Complex Instruction Set Computer (eigentlich Prozessor)



# Mikroprogrammierung

Beispiele CPUs mit Mikroprogrammsteuerung

- Intel 80486, 80586, Pentium
- Motorola 68000 ... 68040
- PDP-11
  Mikroprogrammierbar (z.B. PDP 11/60)!

Tendenz geht (wieder) zu Prozessoren mit wenigen einfachen Befehlen (RISC) mit geringer oder keiner Mikroprogrammsteuerung.