

Mic-1

- Mic-1
 - Registers
 - Namen
 - Descriptionen
 - MDR: Memory Data Register
 - MAR: Memory Address Register
 - PC: Program Counter
 - MBR: Memory Buffer Register
 - MBRU (*pseudo*): Memory Buffer Register Unsigned
 - SP: Stack Pointer
 - LV: Local Variable
 - CPP: Constant Pool Pointer
 - TOS: Top of Stack
 - OPC: Old Program Counter
 - H: Help Register
 - ALU
 - Hauptspeicher
 - Zyklus
 - Speichern
 - Laden
 - Bytecode Zugriff
 - Mikroinstruktionen
 - Mikroinstruktionen Format
 - Addr (9 Bit)
 - JAM (3 Bit)
 - ALU (8 Bit)
 - C-Bus (9 Bit)
 - Mem (3 Bit)
 - B (4 Bit)
 - Kontrollpfad
 - Ablauf eines Zyklus
 - Δw
 - Δx
 - Δy
 - Δz
 - Steigende Flanke

Registers

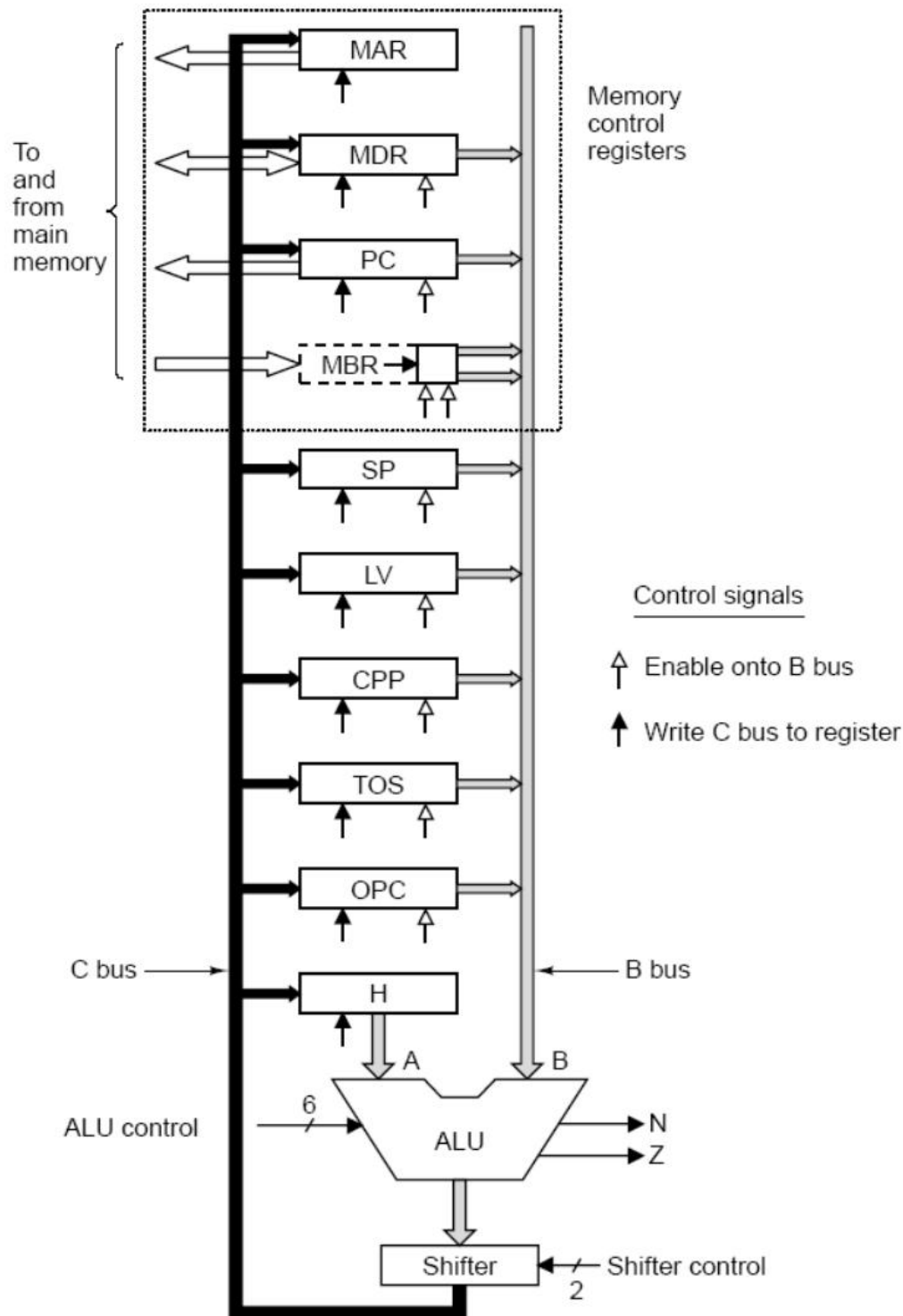


Bild: Structured Computer Organization, 6th Edition, Fig. 4-1

Registers sind im wesentlichen 32-Bit/4-byte Speicher (Ausnahme: **MBR** nur 1 Byte/8 Bit breit).

Namen

Registers mit Speicherzugriff:

- **MDR**: Memory Data Register
- **MAR**: Memory Address Register

- **PC**: Program Counter
 - **MBR**: Memory Buffer Register
-

- **SP**: Stack Pointer
- **LV**: Local Variable
- **CPP**: Constant Pool Pointer
- **TOS**: Top of Stack
- **OPC**: Old Program Counter
- **H**: Help Register

Descriptionen

MDR: Memory Data Register

Beinhaltet das Wert des Speicherwortes, das gelesen wird oder geschrieben werden soll.

MAR: Memory Address Register

Beinhaltet die Adresse des Speicherwortes, das gelesen wird oder geschrieben werden soll.

PC: Program Counter

Beinhaltet die Adresse des nächsten Befehls (in der Method Area). Wird nach jedem Befehl inkrementiert. (in Main1 / 0x100)

MBR: Memory Buffer Register

Beinhaltet das Wert des Speicherwortes, die in Adresse **PC** steht. Also was als nächstes ausgeführt werden soll. (Vorzeichenbehaftet)

MBRU (*pseudo*): Memory Buffer Register Unsigned

Vorzeichenlose Variante von **MBR**. Wird für Vorzeichenlose Operanden verwendet (z.B. **varnum**).

SP: Stack Pointer

Beinhaltet die Adresse des obersten Elements auf dem Stack.

LV: Local Variable

Beinhaltet die Adresse der unteren Rand des aktuellen Stackframes (**OBJREF**). Es wird benutzt, um auf lokale Variablen zuzugreifen. (z.B liegt die 2. lokale Variable in **LV + 2**)

CPP: Constant Pool Pointer

Adresse des ersten Elements im Constant Pool. Es ändert sich nicht während der Laufzeit.

TOS: Top of Stack

Beinhaltet konventionell den Wert des obersten Elements auf dem Stack. dieser Zustand wird nach jedem voreingestellten Mikroprogramm (z. B. bipush) erreicht, muss aber in einem selbst implementierten Mikroprogramm manuell eingestellt werden.

OPC: Old Program Counter

Wird benutzt um verschiedene Werte zwischenspeichern. (z.B. PC bei GOTO / TOS bei if_icmpeq)

H: Help Register

Das verwenden wir, wenn wir zwei Operanten brauchen. Was in H liegt, liegt auch an A an (ALU Eingang).

ALU

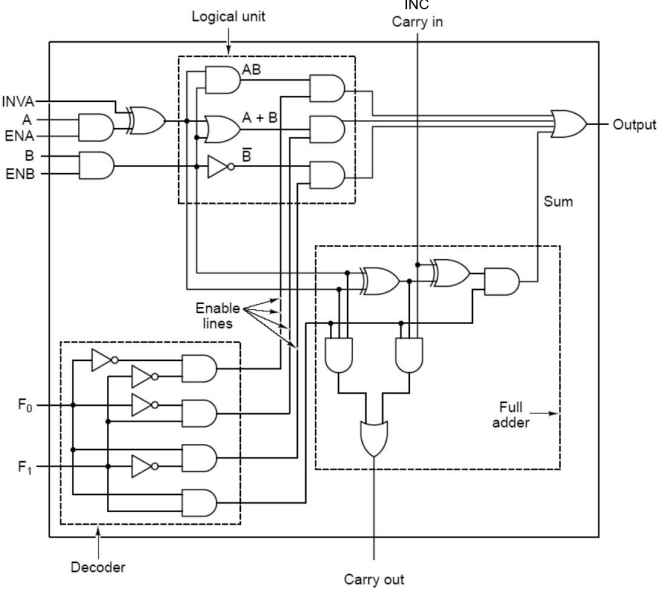


Bild: Structured Computer Organization, 6th Edition, Fig. 3-18

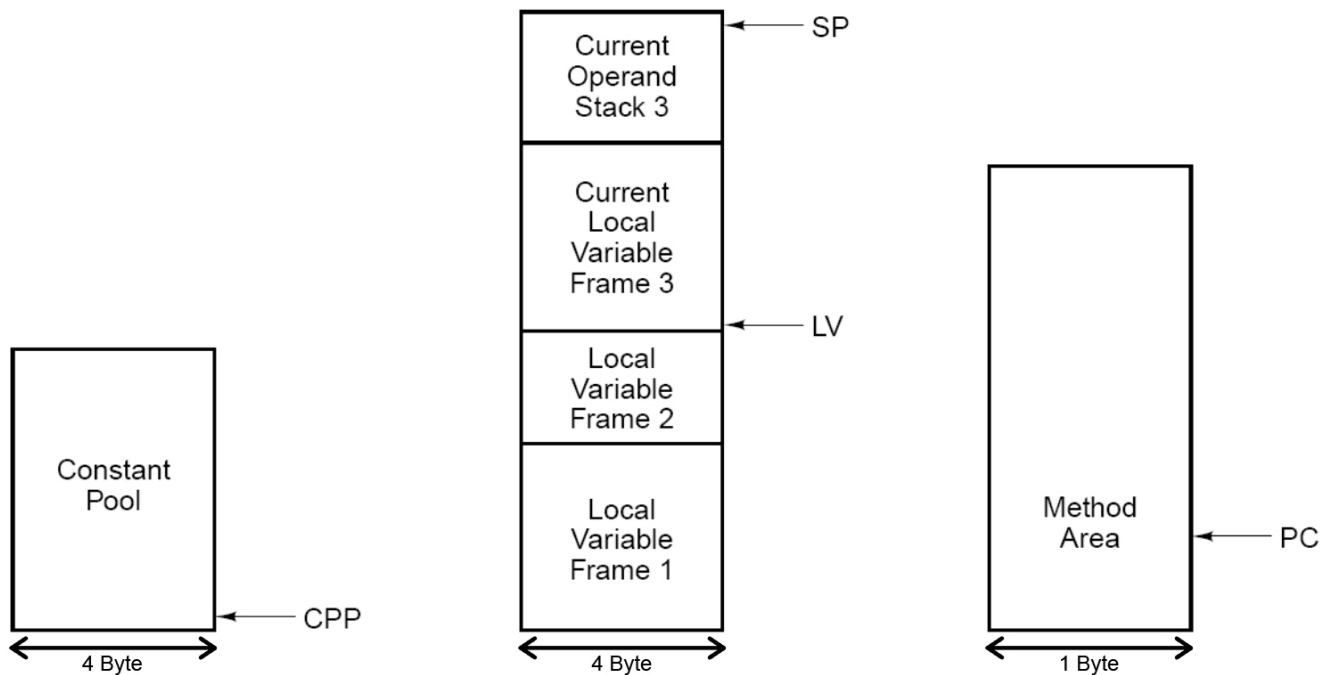
F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	\overline{A}
1	0	1	1	0	0	\overline{B}
1	1	1	1	0	0	$A + B$
1	1	1	1	0	1	$A + B + 1$
1	1	1	0	0	1	$A + 1$
1	1	0	1	0	1	$B + 1$
1	1	1	1	1	1	$B - A$
1	1	0	1	1	0	$B - 1$
1	1	1	0	1	1	$-A$
0	0	1	1	0	0	$A \text{ AND } B$
0	1	1	1	0	0	$A \text{ OR } B$
0	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	-1

Tabelle:: Structured Computer Organization, 6th Edition, Fig. 4-2

Wir haben in der ALU 32 von diese Einheiten in Verkettung.

- **N**: Negative Flag
(Das Ergebnis der letzten Operation eine 1 im MSB hat)
- **Z**: Zero Flag
(Das Ergebnis der letzten Operation eine 0 ist)

Hauptspeicher



Wir haben im Hauptspeicher drei verschiedene Speicherbereiche:

- Constant Pool
- Stack Frame
- Method Area

Wir interagieren (rd, write, fetch) mit dem Hauptspeicher über die **MAR**, **MDR** (für rd und wr) und **PC** (für fetch) **Registers**.

Zyklus

genauer [weiter unten](#)

Speichern

- Lade in **MAR** die Adresse des Wortes, das wir speichern wollen. (1. Zyklus)
- Lade in **MDR** das Wert des Wortes, das wir speichern wollen. (2. Zyklus)
- Speicher signalisieren (Ende des 2. Zyklus)
- Daten sind am Ende des 3. Zyklus im Speicher. **MDR** und **MAR** dürfen während des 3. Zyklus wieder verwendet werden.

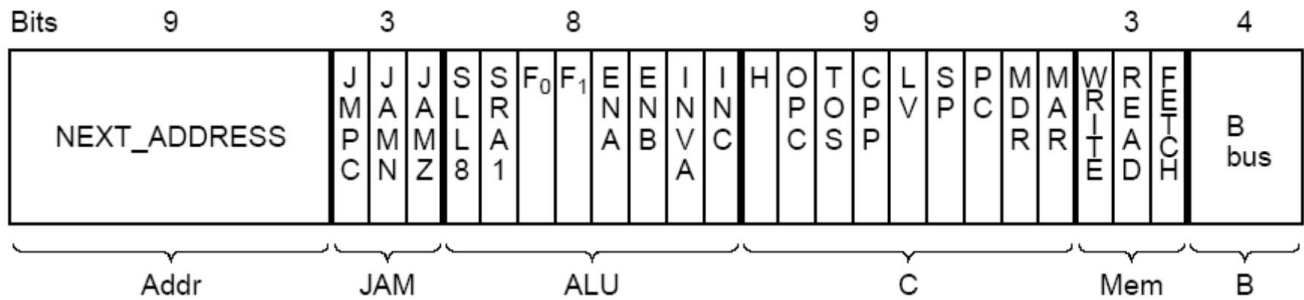
Laden

- Lade in **MAR** die Adresse des Wortes, das wir laden wollen. Signalisiere, dass geladen werden soll (1. Zyklus)
- Ergebnis ist am Ende des 2. Zyklus in **MDR**. Ursprüngliche Inhalt des **MDR** darf im 2. Zyklus noch verwendet werden aber nicht vom C-Bus geschrieben werden (sonst Kollision).
- Ab dem 3. Zyklus darf der neuen Inhalt des **MDR** verwendet werden.

Bytecode Zugriff

Wir nutzen hier die **PC** und **MDR** **Registers**, um den Bytecode bytewise zu lesen, mit einem **Fetch** Signal.

Mikroinstruktionen



Anhand 36-Bit Mikroinstruktionen wird die [ALU](#) und die [Registers](#) gesteuert.

Mikroinstruktionen Format

Addr (9 Bit)

Beinhaltet [NEXT_ADDRESS](#) (Adresse der nächsten Mikroinstruktion), je nachdem was in [JAM](#) steht.

JAM (3 Bit)

- [JMPC](#): Jump on [PC](#)
Bytecode byte wird in [MPC](#) geladen. z.B. [0x60](#) für [IADD](#).
- [JAMN](#): Jump if Negative
 $MPC[8] = NEXT_ADDRESS[8] \text{ OR } N$
- [JAMZ](#): Jump if Zero
 $MPC[8] = NEXT_ADDRESS[8] \text{ OR } Z$

Es gilt insgesamt: $MPC[8] = (JAMZ \text{ AND } Z) \text{ OR } (JAMN \text{ AND } N) \text{ OR } NEXT_ADDRESS[8]$

ALU (8 Bit)

- [SLL8](#): Shift Left Logical 8
- [SRA1](#): Shift Right Arithmetic 1
- [F0](#) - [INC](#): [ALU](#) Input

C-Bus (9 Bit)

Welche Register vom C-Bus geschrieben werden sollen, je nachdem welche Bits auf 1 gesetzt sind.

Mem (3 Bit)

[rd](#), [wr](#) oder [fetch](#) Signal für den Hauptspeicher.

B (4 Bit)

Was auf den B-Bus gelegt werden soll. Dafür ist folgende Kodierung vorgesehen:

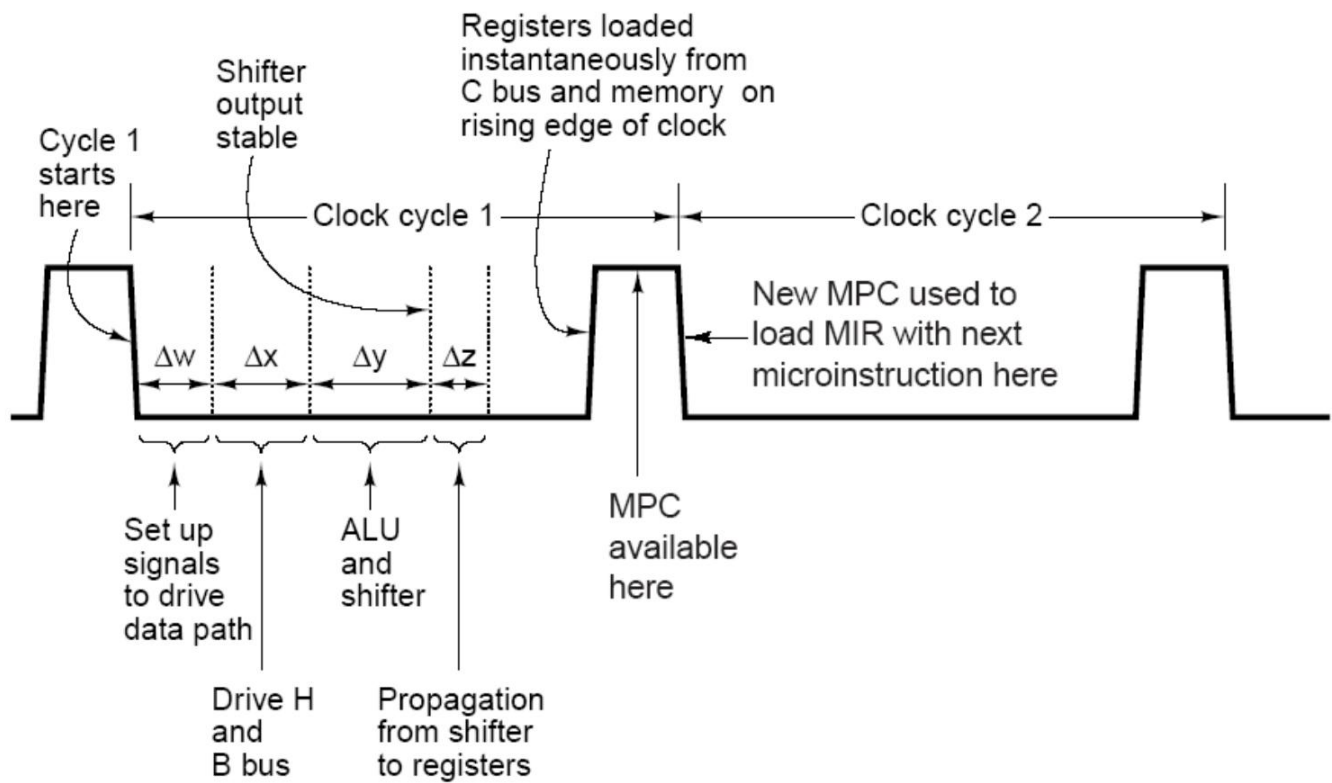


Bild: Structured Computer Organization, 6th Edition, Fig. 4-3

Δw

Wir nehmen die 9-Bit Adresse die in **MPC** steht und laden die Mikroinstruction aus **control store**, die in dieser Adresse steht, in **MIR**.

Δx

- Wert von **B** in **MIR** wird dekodiert und auf den B-Bus gelegt.
- Es wird das Wert vom jeweiligen Register am **ALU** Eingang **B** gelegt.
- Wert von **H** wird am **ALU** Eingang **A** gelegt.

Δy

- **ALU** rechnet gemäß **Mikroinstruction** und leitet das Ergebnis an den Shifter weiter.
- **Shifter** modifiziert das Ergebnis.

Δz

- Ergebnis vom Shifter auf den **C-Bus** stabilisiert sich.

Steigende Flanke

- Register werden vom **C-Bus** neugeladen.