Fortsetzung Speicherbausteine

SRAM (statisches RAM):

Funktionsweise:

1. Schreiben
   - Zu speichender Wert wird an BL bzw. $\overline{BL}$ angelegt
   - Wortleitung wird aktiviert
     ➔ Tortransistoren werden durch gesteuert (T_3, T_4) / leitend
   - Bsp. „1“ ➔ $BL = 5V$, $\overline{BL} = GND$
   - T_1 und T_2 stellen sich entsprechend der anliegenden Gate-Spannung ein ➔ leiten/sperren
     ➔ Der Wert wird gespeichert in dem Zustand der Transistoren T_1 und T_2
   - „1“ ➔ T_1 sperrt, T_2 leitet
   - „0“ ➔ T_2 sperrt, T_1 leitet
   - Wortleitung wird deaktiviert
   - Zustand der Transistoren T_1 und T_2 bleibt erhalten
   - Deaktiviert man die Versorgungsspannung U_B , werden beide Transistoren gesperrt, die Speicherzelle ist dann nicht funktionsfähig

2. Auslesen
   - Gespeichert ist z.B. eine „1“
   - Wortleitung wird aktiviert (mit z.B. 5 V) ➔ Tortransistoren werden durch gesteuert (T_3 und T_4)
   - Werte an den Punkten Q und $\overline{Q}$ (Übergang Tortransistorleitung auf Speichertransistoren) werden auf die Bitleitung gelegt und stehen nun dort zur Verfügung

SRAM – Eigenschaften:

- Sehr geringe Zugriffszeiten ➔ sehr schneller Speicher (Faktor 20 – 30 zu DRAM)
- Nachteil: Im Vergleich zum DRAM mehr Bauteile
  - SRAM: 6 Transistoren (4 Transistoren + 2 Transistoren als Widerstände)
  - DRAM: 1 Transistor, 1 Kondensator

  ➔ Auf gleicher Chipfläche weniger Speicherzellen als beim DRAM

Fortsetzung Rechnerarchitektur DemoCom

- Opcode: Die ersten vier Stellen eines 8-Bit Wortes aus dem Speicher ➔ repräsentiert den Befehl (z.B. 0000 ➔ LDA)
- Objekt: Die zweiten vier Stellen eines 8-Bit Wortes aus dem Speicher ➔ enthält das Argument des Befehls (z.B. die Speicheradresse, deren Wert genutzt werden soll)

Beispielprogramm

- Das Programm soll 2+1-3 berechnen
- Es wird zunächst im Zustand „Programmieren“ in den Speicher geschrieben → eine 1 auf Adresse 5, eine 2 auf 6 und eine 3 auf die 7
- Ablaufcode des Programms:
    - LBA 5 → 1 wird in den Akkumulator geschrieben
    - ADD 6 → 2 wird zum Wert des Akkumulators hinzuaddiert →Ergebnis 3 → wird in Akkumulator geschrieben
    - SUB 7 → 3 wird vom Akkumulator abgezogen → Ergebnis 0 → wird in den Akkumulator geschrieben
    - OUT → der Wert wird auf das LED-Display ausgegeben
    - HLT → das Programm ist beendet, der Prozessor wird gestoppt
- Der Ablauf des Programms wird an die ersten 5 Positionen im Speicher geschrieben → der Programm Counter läuft diese durch, bis er am HLT Befehl stoppt:

  Übersicht des Speichers:

Speicheradresse	Inhalt	Inhalt in HEX-Zahl
0000	LDA 5 (0000 0101)	05
0001	ADD 6 (0001 0110)	16
0010	SUB 7 (0010 0111)	27
0011	OUT (1110 0000)	E0
0100	HLT (1111 0000)	F0
0101	1 (0000 0001)	01
0110	2 (0000 0010)	02
0111	3 (0000 0011)	03

  ➔ Die letzten drei Speicherpositionen würden, wenn sie zum Programm gehören würden, einen Speicheraufruf und ein Laden von Werten in den Akkumulator hervorrufen, die HLT Befehl verhindert dies.

  ➔ 0-4 → Programm, 5-7 Speicher

Ablauf des Programms

- PC (Programm Counter) steht initial auf 0
- Programmbearbeitung (die Zahlen entsprechen dem Taktzyklus), Schritt 1:
    1. LDA Befehl holen
        a. ABL (Ablaufsteuerung) aktiviert E_P, L_M → Inhalt des PC liegt auf dem Bus („0“)
        b. Bei CLK Flanke lädt das MAR den Wert des Busses („0“) → wird an Adressdekoder im RAM weitergegeben → Bitleitungen werden aktiviert, der Wert des Speichers an der Adresse „0“ wird ausgegeben → „05“
    2. Befehl („05“) in das IR (Instruction Register)
        a. Ablaufsteuerung aktiviert C_E, L_I → Auf dem Bus liegt „05“ → wird bei Flanke ins IR geladen

           → Wert des IR: $\underset{Opcode}{0000}\ \underset{Objekt}{0101}$

Opcode gelangt in die ABL (→ diese weiß nun, was als nächstes zu tun ist), Objekt enthält die Speicheradresse des LDA Befehls

3. Objekt-Teil des Befehls wird in MAR geladen
   a. ABL aktiviert E_I, L_M → aus dem RAM wird „01“ über die Bitleitungen bereitgestellt
4. Ram-Inhalt wird in Akkumulator geladen
   a. ABL aktiviert C_E, L_A → Akkumulator erhält den Wert „01“
5. Der PC wird um 1 erhöht
   a. ABL aktiviert C_P → PC enthält dann den Wert „1“

- Programmbearbeitung (die Zahlen entsprechen dem Taktzyklus), Schritt 2:
  1. ADD Befehl holen
     a. ABL aktiviert E_P, L_M → Inhalt des PC liegt auf dem Bus („1“)
     b. Bei CLK Flanke lädt das MAR den Wert des Busses („1“) → wird an Adressdekoder im RAM weitergegeben → Bitleitungen werden aktiviert, der Wert des Speichers an der Adresse „1“ wird ausgegeben
  2. Befehl in das IR
     a. Ablaufsteuerung aktiviert C_E, L_I

     → Wert des IR: $\underset{Opcode}{0001}\ \underset{Objekt}{0110}$

     Opcode gelangt in die ABL (→ diese weiß nun, was als nächstes zu tun ist), Objekt enthält die Speicheradresse des ADD Befehls
  3. Objekt-Teil des Befehls wird in MAR geladen
     a. ABL aktiviert E_I, L_M
  4. Ram-Inhalt wird in Register B geladen
     a. ABL aktiviert C_E, L_B
  5. Register B und Akkumulator werden addiert und das Ergebnis wird in den Akkumulator geschrieben
     a. ABL aktiviert E_U, L_A (ALU rechnet kurz immer direkt den aktuellen Wert aus, entscheidend für das Ergebnis ist der Zeitpunkt der Taktflanke)
  6. Der PC wird um 1 erhöht
     a. ABL aktiviert C_P → PC enthält dann den Wert „2“
- Im Falle einer Subtraktion (Schritt 3) würden sich nur wenige Dinge ändern:
  - RAM-Adresse 0111
  - Akkumulator – Register B → ABL aktiviert E_U, S_U, L_A
- ADD & SUB brauchen 6 Taktzyklen, LDA nur 5
- Der OUT Befehl (Schritt 4) würde folgenden Ablauf haben:
  1. OUT Befehl holen → ABL aktiviert E_P, L_M
  2. Befehl ins IR → ABL aktiviert C_E, L_I
  3. Akkumulator ins Output-Register →ABL aktiviert E_A, L_B

➔ Der Trick ist die ABL, welche die Daten generiert, sie verschiebt und die durchzuführenden Operationen festlegt

➔ Die ABL muss zum Zeitpunkt der Taktflanke die richtigen Befehle ausführen

ABL Kontrollworte

- Diese Werden in 2 Phasen unterschieden:
  - Fetch: Holt den Befehl und zählt den PC um 1 weiter
  - Execute: Ausführen des Befehls mit den entsprechenden Schritten
- Realisiert wird dies durch einen Ringzähler für die Schritte $S1-S6$ und einer Befehlsdekodierung, welche eine Leitung für einen Befehl schaltet (LDA, ADD, SUB, OUT)
- Intern arbeitet der Befehlsdekodierer mit einer logischen Schaltung, welche durch Invertieren des anliegenden Zustandes und entsprechender UND-Verknüpfungen ermittelt, welcher Befehl auszuführen ist
- Die ABL ermittelt mit Hilfe einer Matrix, in der die Schritte $S1-S6$ und die Befehle (LDA, ADD, SUB, OUT) durch logische Verknüpfungen mit den entsprechend zu aktivierenden Leitungen in der Schaltung verbunden sind, wie der Befehl ausgeführt werden muss

Erweiterung zu DemoComPlus

- Bespiel für Schleife:
  I = 5
  Mache etwas ();
  I = I-1 → liegt im Plus als Rechenoperation der ALU vor
  Wenn i nicht 0 → ZRO Flag gibt die Info, ob dieser Zustand vorliegt aus
- PC bekommt eine Verbindung zum Bus und kann damit geladen werden → Sprünge sind möglich
- Speicher wird auf 256 Byte erweitert (8 Adressbits)
- IR bleibt 8 Bit breit → man benötigt weiteres Register, um die Adresse aufnehmen zu können
  ➔ Opcode geht in IR, dann in ABL, danach wird Objekt geladen →wird in Register Z geschoben → reicht es weiter an MUX, dann an MAR
- Moderne Prozessoren kommen ohne Akkumulator aus → sie besitzen mehrere Register, welche auf zwei Argumentbusse in Richtung der Alu geschaltet werden können