Aufgabe 6 *MIMA-Architektur* (7 Punkte)

Die MIMA ist die Ihnen aus der Vorlesung bekannte mikroprogrammierte Minimalmaschine (**siehe Beiblatt: Architektur der MIMA**), die nach dem von-Neumann-Prinzip aufgebaut ist, d. h. Maschinenbefehle werden sequentiell abgearbeitet. In der Lese-Phase wird ein über IAR adressierter Befehl aus dem Speicher gelesen und im IR abgelegt. Die Lese-Phase dauert 5 Taktzyklen. Im 6. Taktzyklus wird der Befehl dekodiert (Dekodier-Phase). Die Ausführungsphase beginnt im 7. Taktzyklus. Nach der Ausführung des Befehls folgt ein Zugriff auf den nächsten Befehl.
Nehmen Sie an, dass ein Hauptspeicherzugriff (Lesen und Schreiben) drei Takte dauert und währenddessen R = 1 bzw. W = 1 sein muss. Eine ALU-Operation sei nach einem Takt abgeschlossen.

Das Mikroprogramm für die Lese-Phase besteht aus fünf Mikrobefehlen:

1. Takt:	IAR → SAR; IAR → X; R = 1	} Lese-Phase
2. Takt:	Eins → Y; R = 1	
3. Takt:	ALU auf Addieren; R = 1	
4. Takt:	Z → IAR	
5. Takt:	SDR → IR	

1. Geben Sie die Mikroprogramme für die Ausführungsphasen der folgenden Maschinenbefehle an (jeweils ab dem 7. Takt, also nach der Lese-Phase und der Dekodier-Phase): 4 P.

   LDC, STV, ADD, EQL

   Beispiel:

   AND:

7. Takt:	IR → SAR; R = 1
8. Takt:	Akku → X; R = 1
9. Takt:	R = 1
10. Takt:	SDR → Y
11. Takt:	ALU auf AND
12. Takt:	Z → Akku

2. Wie viele Operationen kann die ALU der MIMA ausführen? 1 P.

3. Wie viele Befehle können für die MIMA maximal implementiert werden? 1 P.

4. Wie werden aus dem 24-Bit breiten Datenbus die Adressen extrahiert? 1 P.

Aufgabe 7 *MIPS-Assembler* (12 Punkte)

1. Übersetzen Sie die folgenden C-Kontrollstrukturen in MIPS-Assembler. Sie dürfen dabei **nur** die MIPS-Befehle `add, lw, beq, bne` und `j` verwenden.

   Nehmen Sie an, dass die Variablen `i, j` und `k` in den Registern `$s3, $s4` und `$s5` stehen und dass sich die Anfangsadresse von `A` im Register `$s6` befindet. Dabei ist `A` ein Feld aus 32-Bit Integerzahlen. Verwenden Sie die Register `$t0` und `$t1` zur Speicherung temporärer Variablen.

   (a) `do-while`-Schleife: 3 P.

   ```
   do
      i = i + j;
   while ( A[i] == k )
   ```

   (b) `else-if`-Anweisungen: 5 P.

   ```
   if (i == j)
      i = i + A[k];
   else if (i == k)
      i = i + A[j];
   else
      i = j + k;
   ```

2. Übersetzen Sie den folgenden C-Code nach MIPS-Assembler: 4 P.

   (a) `ptr = &i;`

   (b) `i = *ptr;`

   (c) `i = **ptr;`

   (d) `*ptr = i;`

   Dabei gilt: `int i, int *ptr;`

Aufgabe 8 *Pipelining* (6 Punkte)

Das folgende MIPS-Programmstück soll auf einem Prozessor mit DLX-Pipeline ausgeführt werden.

```
S1:    lw    $t1, 0x100($zero)
S2:    lw    $t2, 0x104($zero)
S3:    add   $t3, $t2, $t1
S4:    addi  $t1, $t2, 8
S5:    subi  $t4, $zero, 2
S6:    and   $t5, $t3, $t2
S7:    sw    $t4, 0x100($zero)
S8:    sw    $t5, 0x104($zero)
S9:    sw    $t1, 0x108($zero)
```

1. Bestimmen Sie alle *echten* Datenabhängigkeiten im Programmstück. 4 P.

2. Nehmen Sie an, dass sowohl *load forwarding* als auch *result forwarding* implementiert ist. 2 P.

   Die auftretenden Pipelinekonflikte sollen durch das Einfügen von möglichst wenigen NOP-Befehlen (*No Operation*) behoben werden.

   Ergänzen Sie das obige Programm, so dass es korrekte Ergebnisse liefert. Sie dürfen dabei die Reihenfolge der Befehle **nicht** ändern.

Aufgabe 9 *Cache-Speicher* (12 Punkte)

1. Bei einem Cache-Speicher mit einer Speicherkapazität von 512 KByte ist die Hauptspeicheradresse in ein 16 Bit Tag-Feld, ein 12 Bit Index-Feld und ein 4 Bit Byte-Offset unterteilt. Geben Sie bei der Beantwortung der folgenden Fragen den Lösungsweg an.

   (a) Bestimmen Sie die Blockgröße in Bytes. 1 P.

   (b) Wieviele Einträge besitzt der Cache-Speicher? 1 P.

   (c) Wie ist der Cache-Speicher organisiert? 2 P.

2. Es soll ein 4-fach-assoziativer *(4-way set associative cache)* Cache-Speicher mit 256 Sätzen und einer Blockgröße von 8 Byte realisiert werden. Nehmen Sie an, dass die Hauptspeicheradresse 32 Bit breit ist. Zur Verwaltung eines Cacheblocks wird nur ein Statusbit (*Valid*-Bit: V) verwendet.

   Bestimmen Sie den insgesamt erforderlichen Speicherbedarf zur Realisierung dieses Cache-Speichers. 3 P.

3. Gegeben sei ein direkt-abgebildeter Cache-Speicher (*direct mapped cache*) mit einer Speicherkapazität von 32 Byte und einer Blockgröße von 8 Byte. Als Aktualisierungsstrategie wird ein Durchschreibverfahren *(write throuph policy)* verwendet. Bei dieser Aktualisierungsstrategie wird ein CPU-Datum bei einem *write miss* nur in den Speicher geschrieben. Bei einem *write hit* wird ein CPU-Datum sowohl in den Cache als auch in den Speicher geschrieben.

   Betrachten Sie die folgenden Lese- und Schreibzugriffe auf die in dezimaler Schreibweise angegebenen Adressen:

Adresse	**0**	**16**	**48**	**8**	**56**	**16**	**8**	**56**	**32**	**0**
read/write	r	r	r	r	r	r	r	w	w	r
Index	0	2								
Tag	0	0								
Hit/Miss	Miss									

   Vervollständigen Sie die obige Tabelle im Lösungsblatt. Verwenden Sie dabei **Miss** für Cache-Miss und **Hit** für Cache-Hit. 5 P.

Aufgabe 10 *Speicher* (8 Punkte)

1. Skizieren Sie den Aufbau einer dynamischen RAM-Speicherzelle. 2 P.
2. Wieviele Adressleitungen sind erforderlich bei einem Speicherbaustein mit einer Kapazität von 4096 Bits und einer 512×8-Organisation? Begründen Sie Ihre Antwort. 1 P.
3. Wieviele RAM-Bausteine der Organisation 8k×1 sind notwendig, um einen Speicher mit einer Kapazität von 8k Wörter und einer Wortbreite von 8 Bit zu realisieren? Begründen Sie Ihre Antwort. 1 P.
4. Wie ist ein ROM-Baustein mit der Speicherkapazität von 2048 Bits und 8 Adressleitungen organisiert? Begründen Sie Ihre Antwort. 1 P.
5. Erläutern Sie die folgenden Begriffe:
   (a) RISC-Prozessor. 1 P.
   (b) Direkter Speicherzugriff (DMA). 1 P.
   (c) *Translation-Lookaside-Buffer (TLB).* 1 P.