



Ausgabe: 08. Januar 2024

Abgaben	{	 Theorie	entfällt
		 Praxis	21. Januar 2024
		Rücksprache	22./23. Januar 2024

Aufgabe 1: MIPS Control (2 Punkte)

Für die Realisierung unserer einfachen MIPS-Implementierung haben wir bisher fast ausschließlich Komponenten des Datenpfades entworfen. Für die Steuerung des Prozessors durch den Kontrollpfad ist neben `aluCtrl` aber noch eine weitere Komponente notwendig.

Die Hauptsteuereinheit ist in [1, ab Seite 249] beschrieben. Die Schaltung in Form eines zweistufigen Schaltnetzes ist in Abbildung 1 dargestellt. Implementieren Sie die Hauptsteuereinheit entsprechend der Abbildung in der Architektur `structural` der Datei `mipsCtrl.vhd`. Die Überprüfung der Funktionalität kann durch Verwendung der vorgegebenen Testbench `mipsCtrl_tb` erfolgen.

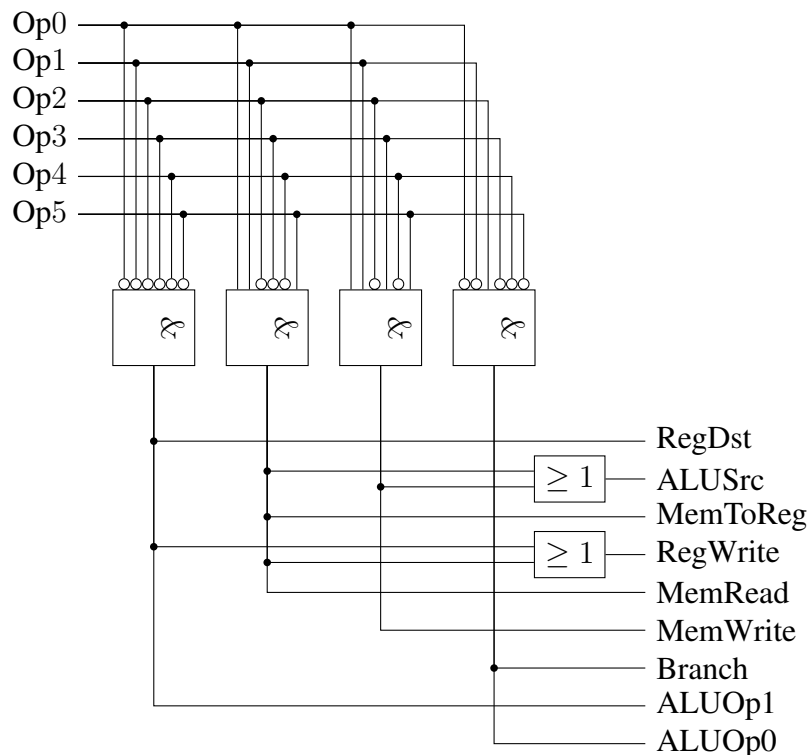


Abbildung 1: Zweistufiges Schaltnetz für die Hauptsteuereinheit aus [1]

Name	Typ	in / out	Beschreibung
op	std_logic_vector (5 downto 0)	in	Opcode der Instruktion
regDst	std_logic	out	Steuersignal für den Multiplexer, der das zu schreibende Register bestimmt
branch	std_logic	out	'1' bei Sprungbefehlen
memRead	std_logic	out	'1' bei Instruktionen, die aus dem Datenspeicher lesen
memToReg	std_logic	out	Steuersignal für den Multiplexer hinter ALU und Datenspeicher
aluOp	std_logic_vector (1 downto 0)	out	gibt die Operation für das ALU-Control-Modul an
memWrite	std_logic	out	'1' bei Instruktionen, die in den Datenspeicher schreiben
aluSrc	std_logic	out	Steuersignal für den Multiplexer, der den zweiten ALU-Operanden auswählt
regWrite	std_logic	out	'1' bei Instruktionen, die ein Register beschreiben

Tabelle 1: MipsControl: Ports

Aufgabe 2: MIPS CPU (18 Punkte)

Sie realisieren in dieser Aufgabe den gesamten MIPS-Prozessor. Verbinden Sie dazu alle notwendigen Komponenten in der Architektur `structural`, wie es in Abbildung 2 dargestellt ist, zu dem aus der Vorlesung bekannten MIPS.

Der Befehlsspeicher wird vorgegeben, um das Laden von Programmen zu ermöglichen. Da der eingeschränkte Befehlssatz keine Operation mit Direktoperanden vorsieht, muss der Datenspeicher mit entsprechenden Werten vorinitialisiert werden, um auf Daten zugreifen zu können. Daher wird der Datenspeicher ebenfalls vorgegeben und nicht Ihre Realisierung verwendet. Funktionell sollte der verwendete Baustein aber identisch zu Ihrer eigenen Implementierung sein.

Die in den Vorgaben enthaltene Datei `mipsCpu.vhd` enthält bereits die Instantiierung der Speichermodule.

Neben der in Abbildung 2 dargestellten Komponenten soll Ihre Implementierung zusätzlich einige Komponenten enthalten, die das Testen der Lösung mithilfe der Testbench ermöglicht (siehe unten). Im Prozessor benötigte Komponenten, bei deren Implementierung nicht die volle Punktzahl erreicht wurde, können aus der `ROrgPrSimLib` verwendet werden. In diesem Fall kann nur Questasim zur Simulation genutzt werden.

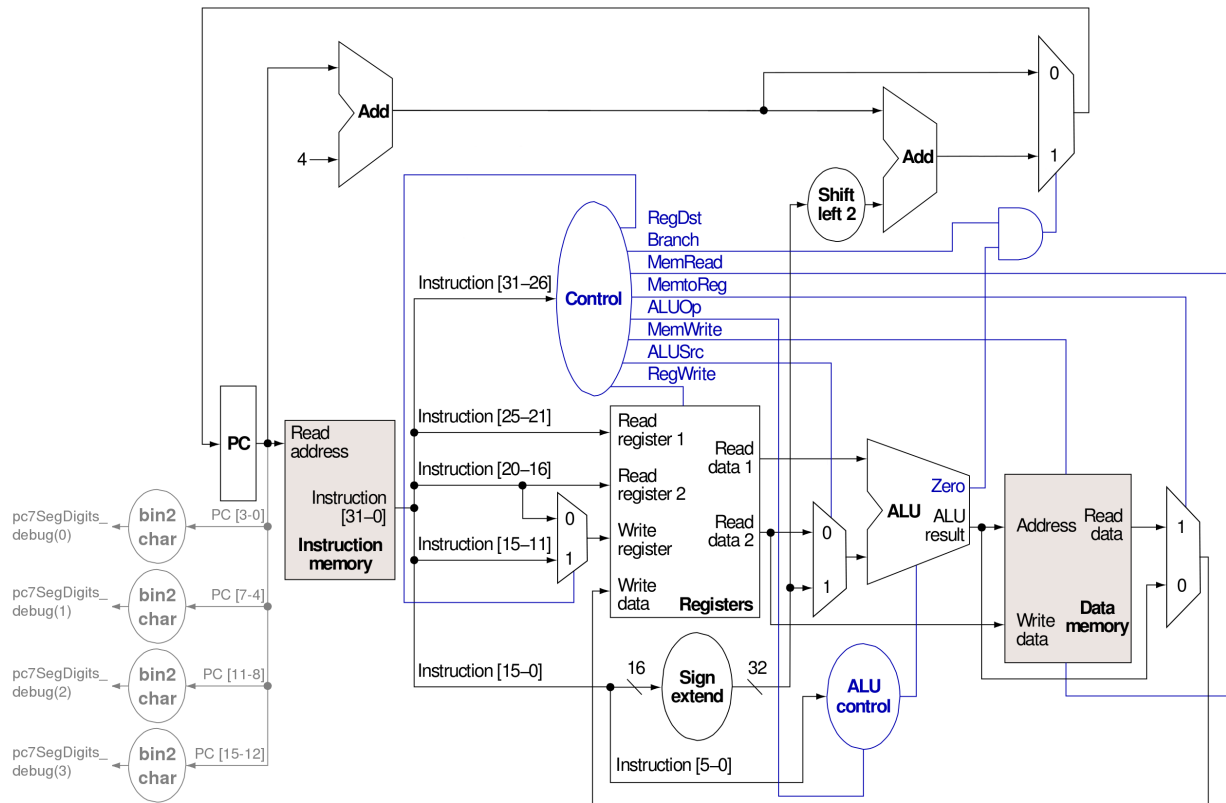


Abbildung 2: Der einfache MIPS mit Steuereinheit, nach [1, S. 252]. Befehls- und Datenspeicher sind gegeben (grau).

Name	Typ	Beschreibung
PROG_FILE_NAME	string	enthält Code-Segment des Testprogramms
DATA_FILE_NAME	string	enthält Daten-Segment des Testprogramms

Tabelle 2: Mips-CPU Generics

Name	Typ	in / out	Beschreibung
clk	std_logic	in	Taktsignal
rst	std_logic	in	Resetsignal

Tabelle 3: Mips-CPU Ports

Beachten Sie bei der Implementierung folgende Hinweise:

- Damit der gesamte Datenpfad in einem Takt durchlaufen werden kann, muss der Datenspeicher mit dem invertierten Taktsignal angesteuert werden.
- Der PC enthält eine Byte-Adresse. Sowohl Instruktions- als auch Datenspeicher werden aber wortweise adressiert.

- Manche Signale verbinden Ports unterschiedlicher Breite miteinander. Überlegen Sie hier, welcher Ausschnitt des längeren Signals verwendet werden muss, bzw. wie das kürzere Signal aufgefüllt werden sollte.
- Verwenden Sie die im `proc_config`-Package definierten Konstanten.

Um den Prozessor vollständig testen zu können, sollen Debug-Funktionalität erweitert werden. Es soll u. A. möglich sein, aus der Testbench einzelne Instruktionen direkt in den Datenpfad einzufügen. Dazu wird ein Multiplexer genutzt, welcher die aus dem Instruktionsspeicher gelesene Instruktion und die per Debug-Port `testInstruction_debug` eingefügte Instruktion als Eingänge erhält. Zur Auswahl dient der Debug-Port `testMode_debug`, welcher angibt, ob der Prozessor im Testmodus betrieben wird. Die ausgewählte Instruktion wird dann im weiteren Datenpfad verwendet. So lange der Prozessor im Testmodus betrieben wird, soll das PC-Register unverändert bleiben. Das gleiche Verhalten soll zum Beschreiben des RAMs implementiert werden. Beachten Sie, dass hier je ein Multiplexer für die Signale `writeEn`, `writeAddr` und `writeData` des RAMs erforderlich ist. Eine vollständige Beschreibung der Debugports finden Sie in Tabelle 4.

Um auf den Inhalt des Registerspeichers zugreifen zu können, muss dessen Debug-Port mit dem entsprechenden Debug-Port der MIPS-CPU verbunden werden (siehe bereits instanziierten RAM zum Vergleich). Verbinden Sie außerdem den nach der Ausführung einer Instruktion aktualisierten PC mit dem `pc_next_debug`-Port. Abschließend müssen die einzelnen 7-Segment-Anzeigen mit dem `pc7SegDigits_debug`-Port verbunden werden.

Sobald sie die Debugfunktionalität implementiert haben, können Sie Ihre Lösung mit der in den Vorgaben enthaltenen Testbench `mipsCpu_tb` testen.

Name	Typ	in / out	Beschreibung
testMode_debug	std_logic	in	bei '1' wird der Prozessor im Testmodus betrieben
testInstruction_debug	std_logic_vector (31 downto 0)	in	Instruktion, die im Testmodus ausgeführt wird
ramInsertMode_debug	std_logic	in	bei '1' kann von außen direkt in den Datenspeicher geschrieben werden (RAM-Insert-Mode)
ramWriteEn_debug	std_logic	in	writeEn im RAM-Insert-Mode
ramWriteAddr_debug	std_logic_vector (LOG2_NUM_RAM_ELEMENTS-1 downto 0)	in	writeAddr im RAM-Insert-Mode
ramWriteData_debug	std_logic_vector (RAM_ELEMENT_WIDTH-1 downto 0)	in	writeData im RAM-Insert-Mode
ramElements_debug	ram_elements_type	out	ermöglicht den Zugriff auf den kompletten RAM-Inhalt
registers_debug	reg_vector_type	out	ermöglicht den Zugriff auf den kompletten Inhalt des Registerspeichers
pc_next_debug	std_logic_vector (PC_WIDTH-1 downto 0)	out	ermöglicht den Zugriff auf den aktualisierten Programmzähler, soll auch im Testmodus auf den theoretisch nächsten Wert des PC gesetzt werden
pc7SegDigits_debug	pc_7seg_digits_type	out	ermöglicht den Zugriff auf die 7-Segment-Anzeigen

Tabelle 4: MIPS-CPU Debugports

Literatur

- [1] David A. Patterson and John L. Hennessy. *Rechnerorganisation und -entwurf*. Spektrum Akademischer Verlag, September 2005.