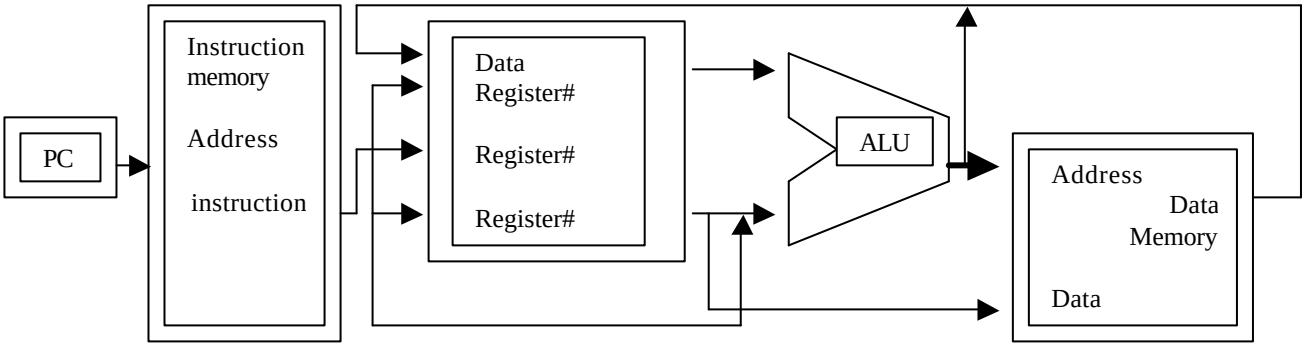
Datapath

Überblick über die Implementation

Um verschiedene Instruktionen, wie MIPS instructions, interger arithmatic-logical instruction und memory-reference instructions zu implementieren muss man für jede dieser Instruktionen

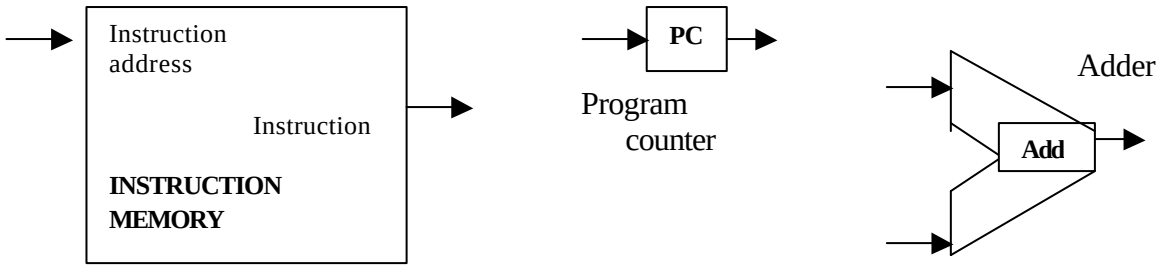
1. Programmbefehl zum Speicher schicken, er muss den Code und die Speicheradresse enthalten
2. Register lesen, meistens zwei, nur bei load word instruction muss ein Register gelesen werden.

Die Beendung der Instruktionen hängt von der Instruktionsklasse (memory-reference, arithmatic- logical, branches) ab.
Die Instruktionsklassen haben Ähnlichkeiten denn alle benutzen die ALU.
Die memory-reference instructions benutzt die ALU für Adressberechnung und die arithmatic-logical Instruktion für Operations Ausführung.
Dannach greift die memory-reference Instruktion auf den Speicher zu um zu speichern oder zu lesen. Die arithmatic-logical Instruktion schreibt Daten von der Alu in die Register.

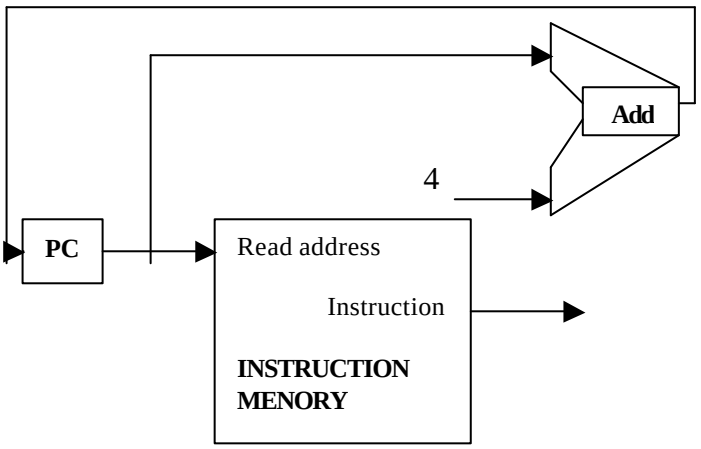


Datenpfad aufbauen

Das erste Element, das man braucht ist ein Platz wo man Programm-Instruktionen speichern kann. Diese Speichereinheit ist ein Zustandselement und wird gebraucht um die Instruktionen zu liefern.
Die Adresse von Instruktionen wird in den program counter (PC) gespeichert. Es ist auch eine state-machine.
Der Addierer (adder),der ist aus einer ALU gebaut. Er inkrementiert den PC zur der Adresse der nächsten Instruktion.



Um eine Instruktion auszuführen, muss man zuerst die Instruktion aus den Instruction-memory holen. Um die nächste Instruktion vorzubereiten, muss man den PC incrementieren,so dass er auf die nächste Instruktion zeigt.

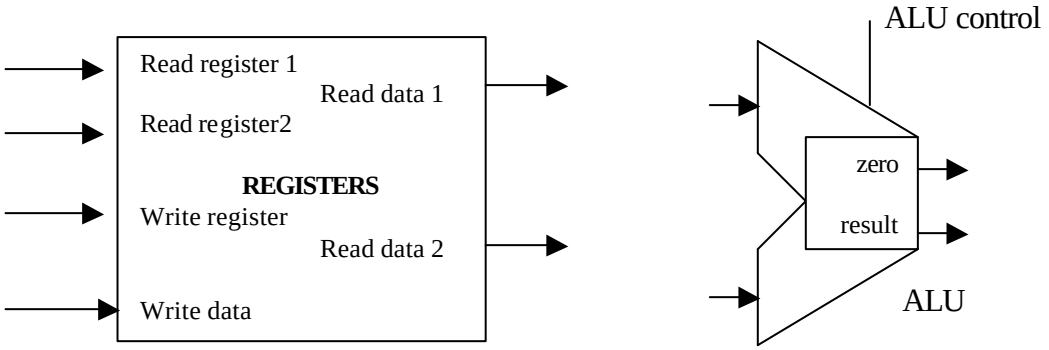


R-format instructions

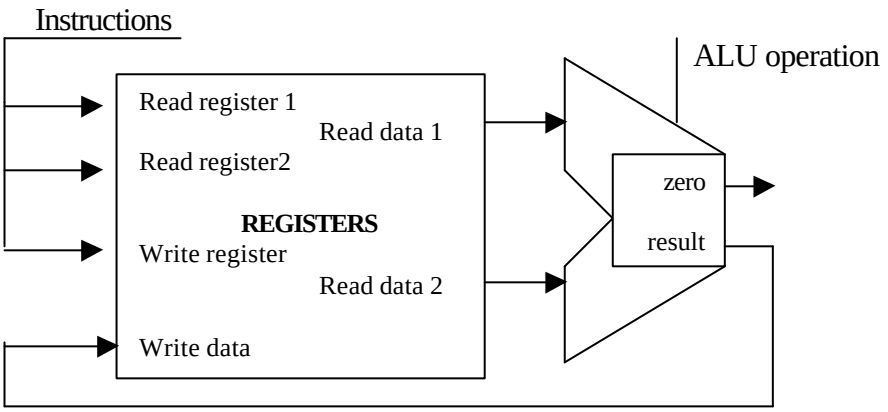
Das ist der Name von einem arithmetischen instuctions format.
Diese Instruktionen lesen 2 Register, führen eine ALU Operation aus und schreiben das Ergibniss. Diese Arithmetic-logic instruction class enthält add, sub, slt and, or.
Die 32 Prozessor register sind in einer Struktur gespeichert, genannt register file. Register file ist einen Sammlung von Registern inder geschrieben und gelesen werden kann. Man braucht eine ALU um die Werte der Register zu lesen. Da R-format instructions 3 Register Operanden haben, braucht man zwei read register um zwei Wörter lesen zu können und einen write Register.
Um ein data-word zu schreiben braucht man noch ein zweiten Register indem die Daten gespeichert werden.
Die write-register werden von den write-control signal gesteuert.
Insgesamt braucht man 4 Inputs (3 für Register Nummer und ein für Daten) und zwei Outputs(beides data Ausgänge).

Die register number Eingänge sind 5 bit weit.

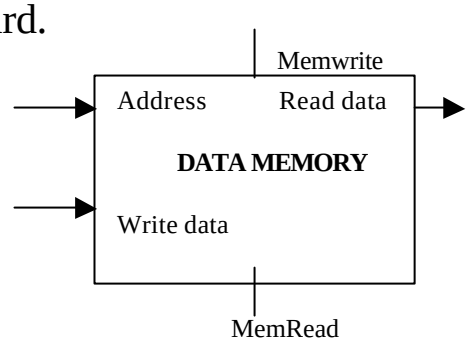
ALU
Die ALU wird gesteuert von einem 3 Bit Signal. Sie nimmt 2*32 Bit an und gibt ein 32 Bit Ergibniss aus.

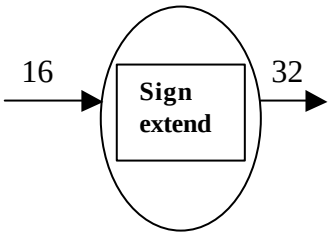


Datapath für R-type instructions



Diese Einheit braucht man um die Befehle laden, speichern zu Implementieren .
Die Data Memory Unit ist eine Zustandsmaschine mit Eingängen für die Adresse und Daten. Und ein Ausgang für das Ergibiss lesen. Es gibt unabhängige control Eingänge für das Schreiben und Lesen. Es kann aber nur ein Signal bei einem clock bearbeitet werden. Die Sign-extension Unit hat einen16 Bit Eingang der in ein 32 Bit gewandelt wird.

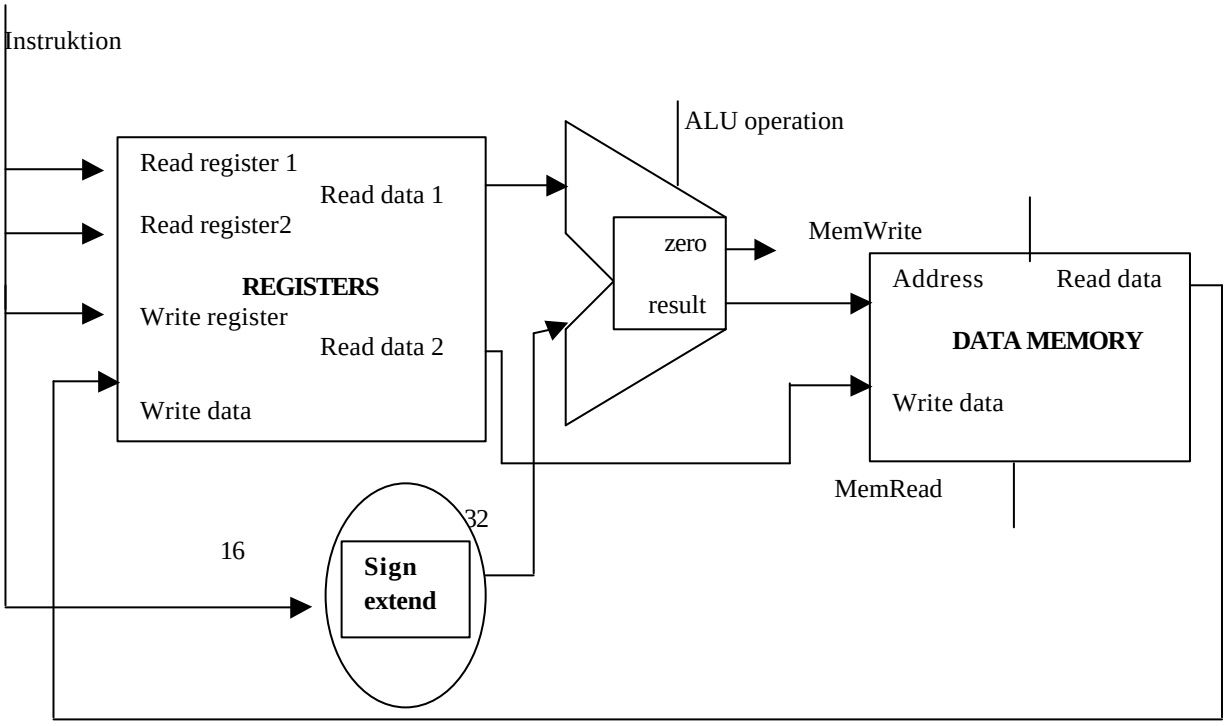




Sign-extension unit

Datapath bestehend aus vorherigen Elementen.
Angenommen, dass die Instruktionen schon geholt sind

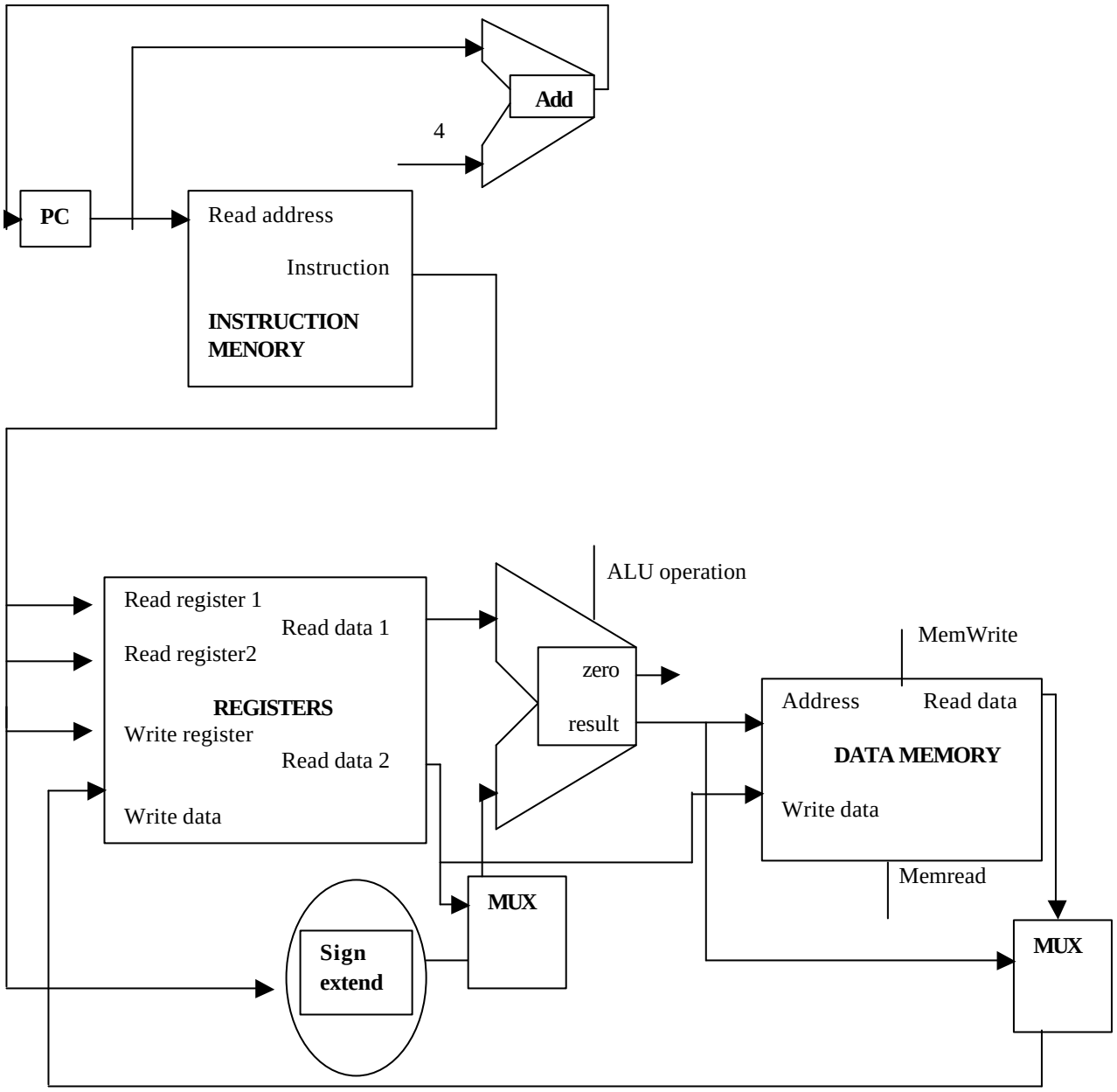
?? die register-number-inputs kommen von den geholten instructions
?? der zweite Eingang bekommt den Wert, der von der Sign-extension-unit erweitert wurde
?? dann berechnet die ALU die Speicheradresse
?? es wird entweder im Speicher geschrieben oder gelesen
?? schlisslich werden die Daten ins Register geladen wenn die nächste Instruktion geladen wird.



Datapath Implementation

Dieser Datapath führt alle Instruktionen in einem clock cyclus aus, d.h. die Datapath Resourcen können nicht mehr als einmal pro Instruktion gebraucht werden. Wenn ein Element zweimal benutzt werden muss, dann muss er verdoppelt werden.
Neu hinzugekommenist der Multiplexor, es ist ein Bauteil,dass dem Ausgang abhängig von den Controlsignal wählt.

PC und instruction-memory:Bei diesem Teil wird die Instruktion geholt und der PC (program counter) incrementiert. Die add-ALU incrementiert den PC während die Haupt-ALU die instruction ausführt.



Mikroprozessoren:

The Processor: Datapath and Control

A Multicycle Implementation

Das Ziel einer Multizyklus Implementierung eines Datenpfads ist es durch die Zerlegung der Befehle in einzelne Schritte und die nacheinander Ausführung dieser Schritte Hardware einzusparen, da die funktionellen Einheiten (Speicher; Register, ALU) während eines Befehls mehrfach genutzt werden können sofern sie in unterschiedlichen Zyklen liegen, und zu erreichen, dass Befehle unterschiedlich lange Dauern können (bei der single cycle implementation dauern alle Befehle so lange wie alle Schritte des längsten Befehls zusammen). Dadurch benötigt man im Unterschied zur single cycle implementation nur eine Speichereinheit und nur eine ALU und es werden einige Register hinzugefügt, um Zwischenergebnisse aufzunehmen, die während eines späteren Zyklus benötigt werden. Die Register, die hierfür benötigt werden sind: das Instruction register (IR) und das Memory data register (MDR), welche die Ausgabe des Speichers zwischenspeichern (Befehl bzw. Daten), die Register A und B, die die Ausgabe des Hauptregisters zwischenspeichern und das ALUOut register, das die Ausgabe der ALU zwischenspeichert. Außer dem Instruction register, das seinen Inhalt bis zum Ende des Befehls halten muss, brauchen diese Register ihren Inhalt immer nur bis zum nächsten Zyklus zu speichern.

Da nun eine ALU alle Daten erhalten muss, die bei einer single cycle implementation an mehrere gingen, müssen nun zusätzlich für den ersten ALU Eingang ein Multiplexor, der zwischen den Ausgaben des Registers A und des Program Counters (PC) wählt, und für den zweiten ALU Eingang ein Multiplexor, der zwischen der Ausgabe des Registers B, der Konstante 4, dem sign-extended Feld und dem shifted offset Feld wählt.

Durch die oben genannten Änderungen im Vergleich zur single cycle implementation werden ein Speicher und zwei Addierer gespart und durch kleinere Teile (Register, Multiplexor) ersetzt, was insgesamt zu einer Verringerung der Hardwarekosten führt.

Da dieser Datenpfad mehrere Zyklen zum Ausführen eines Befehls benötigt, muss nun noch eine Kotrolleinheit hinzugefügt werden, die bestimmt wann welches Register mit welchem Wert beschrieben werden soll.

Kontrollsignale:

Actions of the 1-bit control signals

Signal name	***Effect when deasserted***	***Effect when asserted***
RegDst	The register file destination number for the Write register comes from the rt field.	The register file destination number for the Write register comes from the rd field.
RegWrite	None	The general-purpose register selected by the Write register number is written with the value of the Write data input.
ALUSrcA	The first ALU operand is the PC.	The first ALU operand comes from the A register.
MemRead	None	Contents of memory at the location specified by the Address input is put on Memory data

		output.
MemWrite	None	Memory contents at the location specified by the Address input is replaced by value on Write data input.
MemtoReg	The value fed to the register file Write data input comes from ALUOut.	The value fed to the register file Write data input comes from the MDR.
IorD	The PC is used to supply the address to the memory unit.	ALUOut is used to supply the address to the memory unit.
IRWrite	None	The output of the memory is written into the IR.
PCWrite	None	The PC is written; the source is controlled by PCSource.
PCWriteCond	None	The PC is written if the Zero output from the ALU is also active.

Actions of 2-bit control signals

Signal name	***Value***	***Effect***
ALUOp	00	The ALU performs an add operation.
	01	The ALU performs a subtract operation.
	10	The funct field of the instruction determines the ALU operation.
ALUSrcB	00	The second input to the ALU comes from the B register.
	01	The second input to the ALU is the constant 4.
	10	The second input to the ALU is the sign-extended, lower 16 bits of the IR.
	11	The second input to the ALU is the sign-extended, lower 16 bits of the IR shifted left 2 bits.
PCSource	00	Output of the ALU (PC + 4) is sent to the PC for writing.
	01	The contents of ALUOut (the branch target address) are sent to the PC for writing.
	10	The jump target address (IR[25-0] shifted left 2 bits and concatenated (verknüpft) with PC + 4[31-28]) is sent to the PC for writing.

Zerlegung der Befehle in Zyklen

Die Befehle sollen nun so zerlegt werden, dass die Operationen, die verrichtet werden, möglichst gleichmäßig über die Zyklen verteilt werden, wodurch dann auch der Zyklus kurz gehalten werden kann. So soll ein Zyklus z. B. höchstens eine ALU Operation oder einen Speicherzugriff oder einen Hauptregisterzugriff enthalten, wodurch der Zyklus nur so lang wie die längste dieser Operationen wird und gleichzeitig festgelegt wird was in einen Zyklus passt. Hierbei geschehen alle Operationen während eines Zyklus gleichzeitig ausgeführt und die Zyklen zur Ausführung eines Befehls nacheinander.

Die ersten beiden Schritte sind für alle Befehle gleich. Da noch nicht bekannt ist um welchen Befehl es sich handelt, können nur Operationen ausgeführt werden, die bei allen Befehlen identisch sind oder die, wenn sie nicht von dem entsprechenden Befehl benötigt werden, einfach ignoriert werden können. Der Vorteil diese Operationen früh auszuführen liegt darin, dass dann die Ergebnisse schon bereit liegen, falls sie gebraucht werden sollten. Die Schritte zur Ausführung eines Befehls sind:

1. Instruction fetch step: IR = Memory[PC];

PC = PC + 4

Der Befehl wird aus dem Speicher gelesen und in das Instruction register geschrieben, der Programm Counter wird erhöht.

2. Instruction decode and register fetch step: A = Reg[IR[25-21]];

B = Reg[IR[20-16]];

ALUOut = PC + (sign-extend (IR[15-10]) << 2);

Das Register wird gelesen und die Werte werden in A und B gespeichert, die Zieladresse des Branch Befehls wird berechnet (Ergebnis kann ignoriert werden, falls es kein Branch Befehl ist) und in den ALUOut gespeichert, wo sie im nächsten Zyklus abgerufen werden können, falls dies nötig wird.

3. Execution, memory address computation, or branch completion: Dies ist der erste Schritt bei

dem die Operationen von der Art des Befehls abhängen:

Memory reference: ALUOut = A + sign-extend (IR[15-0]);

Die ALU berechnet die Adresse für einen Speicherzugriff aus A und der Ausgabe der sign extension unit und speichert das Ergebnis im ALUOut.

Arithmetic-logical instruction (R-type): ALUOut = A op B;

Die ALU führt die arithmetisch-logische Operation, die im Befehl angegeben ist, mit den Werten aus A und B (aus Schritt 2.) aus und speichert das Ergebnis im ALUOut.

Branch: if (A == B) PC = ALUOut;

Die ALU überprüft A und B auf Gleichheit (falls A und B gleich sind wird die Branchzieladresse aus Schritt 2. die Adresse für den nächsten Befehl).

Jump: PC =PC [31-28] || (IR[25-0]<<2)

Der PC wird mit der Jump Adresse ersetzt.

4. Memory access or R-type instruction completion step:

Memory reference: Laden: MDR = Memory [ALUOut];

oder

Speichern: MDR [ALUOut] = B;

Bei der Operation Laden werden Daten aus dem Speicher in das Memory data register geschrieben. Bei der Operation Speichern werden Daten in den Speicher geschrieben. In beiden Fällen wird die Adresse aus Schritt 3. benutzt.

Arithmetic-logical instruction: Reg[IR[15-11]] = ALUOut;

Das Ergebnis aus dem 3. Schritt wird ins Register geschrieben.

5. Memory read completion step: Reg[IR[20-16]] = MDR;

   Daten, die im 4. Schritt in das Memory data register geschrieben wurden, werden in das Hauptregister geschrieben.

Schritte zur Ausführung eines Befehls:

Step name	***Action for R-Type instructions***	***Action for memory reference instructions***	***Action for branches***	***Action for jumps***
Instruction fetch	IR = Memory[PC] PC = PC + 4			
Instruction decode/ register fetch	A = Reg [IR[25-21]] B = Reg[IR[20-16]] ALUOut = PC + (sign-extend (IR[15-0]) << 2)			
Execution, address computation, branch/jump completion	ALUOut = A op B	ALUOut = A + sign-extended(IR[15-0])	if (A == B) then PC = ALUOut	PC = PC [31-28] || (IR [25-0]<<2)
Memory access or R-type completion	Reg[IR[15-11]] =ALUOut	Load: MDR = Memory[ALUOut] or Store: Memory [ALUOut] = B		
Memory read completion		Load: Reg[IR[20-16]] = MDR		

Definieren der Kontrolleinheit

Die Kontrolleinheit für den Multicycle Datapath muss die Operationen eines Schrittes festlegen und anzeigen, welcher Schritt dann folgt. Es gibt zwei Methoden, um dies zu erreichen:

Die erste Methode ist die Finite state machine. Diese besteht aus einer Anzahl von Zuständen und Richtungen für Zustandsänderungen (grafische Darstellung). Die Richtung für eine Zustandsänderung wird aus dem gegenwärtigen Status und durch Eingaben, die folgen müssen um zu einem anderen Status zu gelangen, gegeben. Die ersten beiden Schritte der Finite state machine sind für alle Befehle gleich,danach hängen die Schritte vom auszuführenden Befehl ab. Nach dem letzten Schritt eines Befehls beginnt die Finite state machine wieder von vorne. Der Prozeß je nach Befehl einen anderen Status auszuwählen nennt sich decoding.

Die zweite Methode ist die des Microprogramming. Da die Finite state machine für den kompletten Befehlssatz des MIPS zu komplex und zu fehleranfällig wird, wird ein kleines Programm mit sogenannten microinstructions geschrieben, das festlegt, welche Kontrollsignale auf welche Weise geschaltet werden und welche microinstruction als nächstes ausgeführt wird. Das Mikroprogramm wird möglichst einfach gehalten, um es besser schreiben und lesen zu können und um zu verhindern, dass fehlerhafte microinstructions entstehen. Eine microinstruction besteht aus sieben Feldern, von

denen die ersten sechs die Kontrollfelder sind und das siebte angibt welche microinstruction als nächstes ausgeführt wird. Um die nächste microinstruction auszuwählen gibt es drei Möglichkeiten: 1. die Ausführung als Sequenz, es folgen die nächsten microinstructions des Mikroprogramms der Reihenfolge nach (häufig Standardeinstellung); 2. zu der microinstruction gehen, die beginnt den nächsten Befehl auszuführen (Fetch); 3. die nächste microinstruction wird anhand der Eingaben in die Kontrolleinheit und einer Tabelle (dispatch table), die die dazugehörigen folgenden microinstructions enthält, ausgewählt. Ein Mikroprogramm kann als eine Textrepräsentation einer Finite state machine angesehen werden.

Die sieben Felder einer microinstruction:

Field name	***Function of Field***
ALU Control	Specify the operation being done by the ALU during this clock; the result is always written in ALUOut.
SRC1	Specify the source for the first ALU operand.
SRC2	Specify the source for the second ALU operand.
Register control	Specify read or write for the register file, and the source of the value for a write.
Memory	Specify read or write, and the source for the memory. For a read, specify the destination register.
PCWrite control	Specify the writing of the PC.
Sequencing	Specify how to choose the next microinstruction to be executed.

Werte, die ein Feld einer microinstruction annehmen kann:

Field name	***Values for field***	***Function of field with specific value***
Label	Any string	Used to specify labels to control microcode sequencing. Labels that end in a 1 or 2 are used for dispatching with a jump table that is indexed based on the opcode. Other labels are used as direct targets in the microinstruction sequencing. Labels do not generate control signals directly but are used to define the contents of dispatch tables and generate control for the Sequencing field.
ALU control	Add	Cause the ALU to add.
	Subt	Cause the ALU to subtract; this implements the compare for branches.
	Func code	Use the instruction´s funct field to determine the ALU control.
SRC1	PC	Use the PC as the first ALU input.
	A	Register A as the first ALU input.
SRC2	B	Register B as the second ALU input.
	4	Use 4 for the second ALU input.
	Extend	Use output of the sign extension unit as the second ALU input.
	Extshft	Use the output of the shift-by-two unit as the second ALU input.
Register control	Read	Read two registers using the rs and rt fields of the IR as the register numbers, putting the data into registers A and B.
	Write ALU	Write the register file using the rd field of the IR as the register number and the contents of ALUOut as the data.
	Write MDR	Write the register file using the rt field of the IR as the register number and the contents of the MDR as the data.

Memory control	Read PC	Read memory using the PC as address; write result into IR (and the MDR).
	Read ALU	Read memory using the ALUOut as address; write result into MDR.
	Write ALU	Write memory using the ALUOut as address; contents of B as the data.
PCWrite control	ALU	Write the output of the ALU int the PC.
	ALUOut-cond	If the Zero output of the ALU is active, write the PC with the contents of the register ALUOut.
	Jump address	Write the PC with the jump address from the instruction.
Sequencing	Seq	Choose the next microinstruction sequentially.
	Fetch	Go to the first microinstruction to begin a new instruction.
	Dispatch i	Dispatch using the ROM specified by i (1 or 2).

Das Mikroprogramm für die Kontrolleinheit:

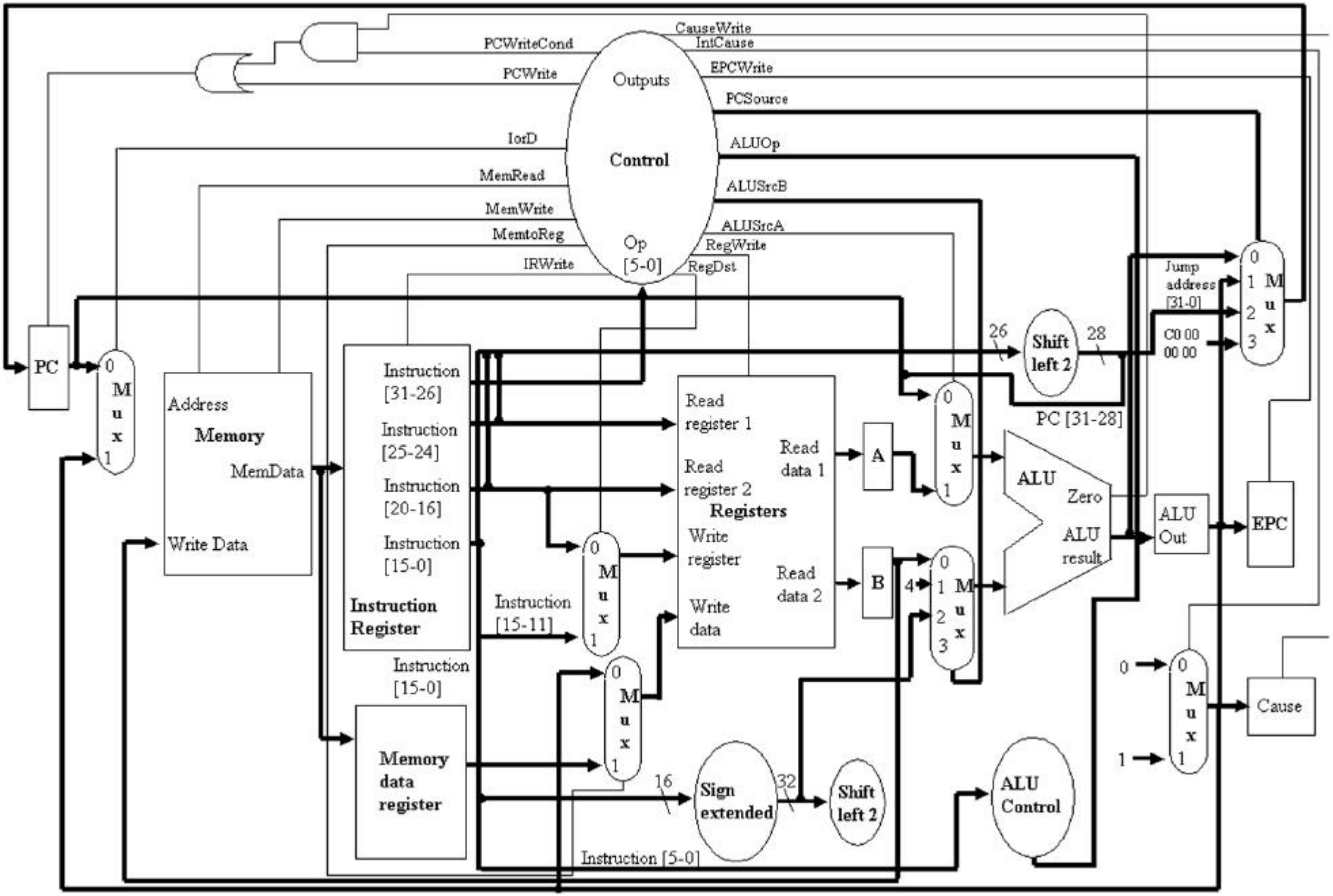
Label	***ALU Control***	***SRC1***	***SRC2***	***Register control***	***Memory***	***PCWrite control***	***Sequencing***
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	PC	Extshft	Read			Dispatch 1
Mem1	Add	A	Extend				Dispatch 2
LW2					Read ALU		Seq
				Write MDR			Fetch
SW2					Write ALU		Fetch
Rformat1	Func code	A	B				Seq
				Write ALU			Fetch
BEQ1	Subt	A	B			ALUOut-cond	Fetch
JUMP1						Jump address	Fetch

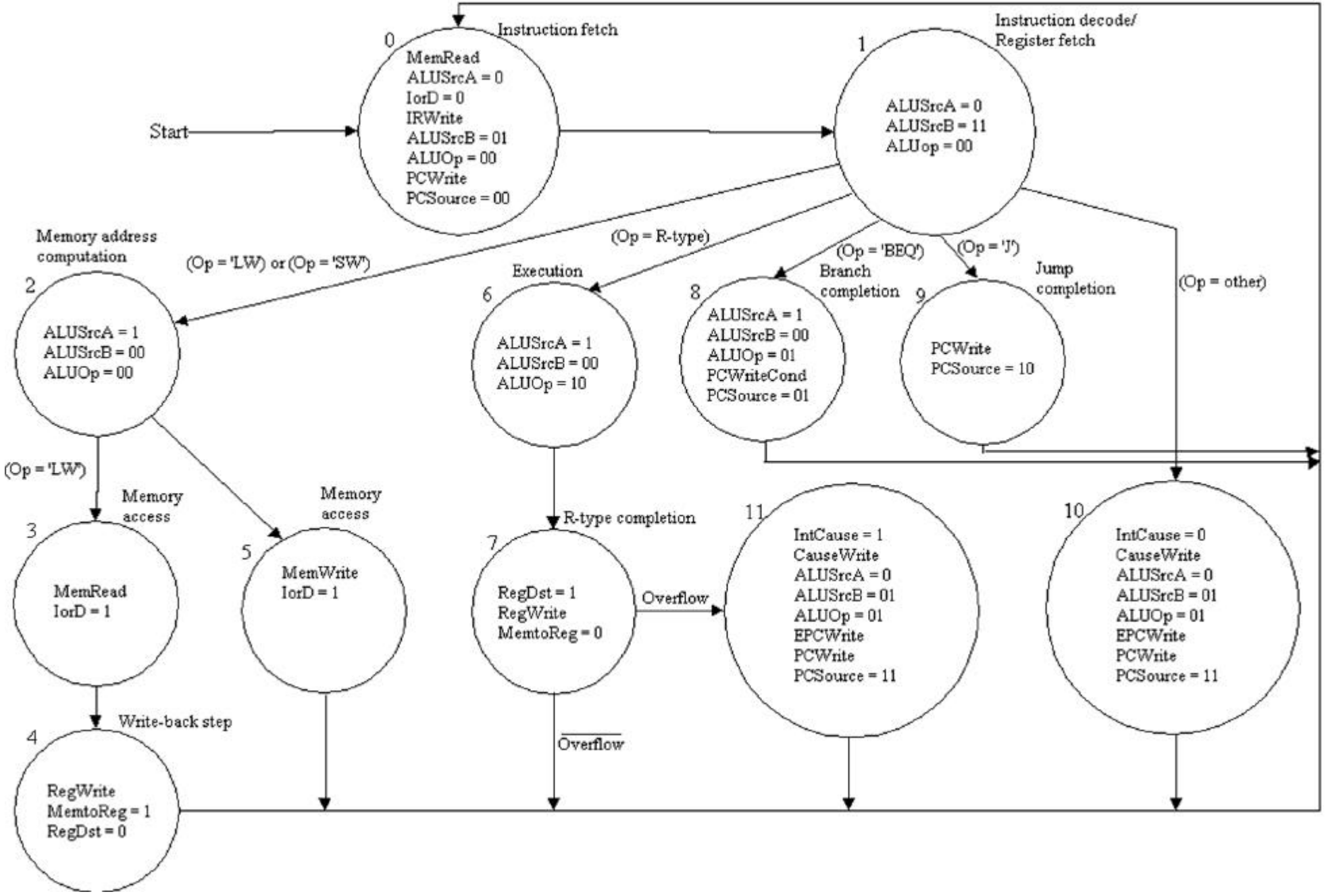
Label wird dazu benutzt eine microinstruction als Ziel einer anderen microinstruction anzugeben (Name einer microinstruction).

Die Schwierigkeit beim erstellen einer Kontrolleinheit ist es die Kontrolle klein und schnell zu halten und trotzdem alle Möglichkeiten zu berücksichtigen.

Ausnahmefehler/Exception

Eine Exception ist irgendeine unerwartete Veränderung in der Abarbeitung der Kontrolleinheit. Wenn dieser Fall eintritt, was passieren kann, wenn für einen Status kein Folgestatus definiert ist oder wenn ein arithmetischer Überfluss (Arithmetic Overflow) bei der ALU auftritt,wird die Adresse, die den Fehler ausgelöst hat im Exception program counter (EPC) gespeichert und dann die Kontrolle an das Betriebsystem übergeben. Dieses führt dann für diesen Fall vordefinierte Operationen durch und danach wird das Programm entweder beendet oder an der vorher gespeicherten Stelle fortgeführt.


CauseWrite
PCWriteCond
IntCause
PCWrite
Outputs
EPCWrite
PCSource
IorD
Control
ALUOp
MemRead
ALUSrcB
MemWrite
ALUSrcA
MemtoReg
Op
[5-0]
RegWrite
IRWrite
RegDst
Jump
address
[31-0]
26
Shift
left 2
28
C0 00
00 00
PC
PC [31-28]
Address
Memory
MemData
Write Data
Instruction
[31-26]
Instruction
[25-24]
Instruction
[20-16]
Instruction
[15-0]
Instruction
Register
Instruction
[15-11]
Instruction
[15-0]
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
A
B
4
ALU
Zero
ALU
result
ALU
Out
EPC
Memory
data
register
16
Sign
extended
32
Shift
left 2
ALU
Control
Instruction [5-0]
0
1
Cause
M
u
x


Start
0
Instruction fetch
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
1
Instruction decode/
Register fetch
ALUSrcA = 0
ALUSrcB = 11
ALUop = 00
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
(Op = other)
2
Memory address
computation
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 00
6
Execution
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
8
Branch
completion
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
9
Jump
completion
PCWrite
PCSource = 10
(Op = 'LW')
3
Memory
access
MemRead
IorD = 1
5
Memory
access
MemWrite
IorD = 1
7
R-type completion
RegDst = 1
RegWrite
MemtoReg = 0
Overflow
11
IntCause = 1
CauseWrite
ALUSrcA = 0
ALUSrcB = 01
ALUOp = 01
EPCWrite
PCWrite
PCSource = 11
10
IntCause = 0
CauseWrite
ALUSrcA = 0
ALUSrcB = 01
ALUOp = 01
EPCWrite
PCWrite
PCSource = 11
4
Write-back step
RegWrite
MemtoReg = 1
RegDst = 0
Overflow