Computer Architecture Project Report

Rasoul Akhavan Mahdavi, Ali Asgari

Special Instructions

Some Instructions in the ISA require more than one cycle to be executed completely(Forget the case of hazards). These involve those instructions that

- Need two Memory accesses(Add Indirect)
- Need two WriteBacks(Multiply)

The solution for these instructions is breaking them into two micro-instructions that each can be done in one clock cycle. To be able to handle the stated instructions, we will introduce new instructions as follows:

Opcode	Operation	Description
10111	Multiply High[1]	R1 = {R2 × R3} [63:32]
11000	Multiply Low[2]	R1 = {R2 × R3} [31:0]
11010	Load Indirect[3]	R1 = Mem[R2]
11001	Single Add Indirect[4]	R1 = Mem[R2] + R3

The decomposition of the stated instructions into smaller instructions are as follows:

[1] Type 1
[2] Type 1
[3] Type 3
[4] Type 1

Original Instruction	Decomposition
MUL R1,R2,R3,R4	MULL R2,R3,R4 MULH R1,R3,R4
ADDNDR R1,R2,R3	LWNDR R1, R2 SADDNDR R1,R3

Extended ISA

The ISA including the added instructions, is as follows:

Type 1 Instructions

Opcode					Reg1					Reg2					Reg3					Reg4					Shift Amount					Unused	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Opcode	Operation	Description
00001	Add	R1 = R2 + R3
00010	Subtract	R1 = R2 − R3
00011	AND (Bitwise)	R1 = R2 & R3
00100	OR (Bitwise)	R1 = R2 | R3
00101	Shift Left Logical	R1 = R2 << Shift Amount
00110	Shift Right Logical	R1 = R2 >> Shift Amount
00111	Max	R1 = Max (R2 , R3)
01000	Set on Less Than	if (R2 < R3) then R1 = 1 , else R1 = 0
01001	Multiply	{R1 , R2} = R3 × R4
01010	Move	R1 = R2
01011	Add Indirect	R1 = Mem[R2] + Mem[R3]

10111	Multiply High	R1 = {R2 × R3} [63:32]
11000	Multiply Low	R1 = {R2 × R3} [31:0]
11001	Single Add Indirect	R1 = Mem[R2] + R3

Type 2 Instructions

Opcode					Reg1					Reg2					Immediate													Unused
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Opcode	Operation	Description
01100	Load Upper Immediate	R1[31:16] = Imm , R1[15:0] = 0x0000
01101	Add Immediate	R1 = R2 + SE(Imm)
01110	OR Immediate	R1 = R2 | Imm
01111	Set on Less Than Immediate	if (R2 < Imm) then R1 = 1 , else R1 = 0

Type 3 Instructions

Opcode					Reg1					Reg2					Address									Unused								
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Opcode	Operation	Description
10000	Branch if Equal	if (R1 == R2) then PC ← Address
10001	Branch if Not Equal	if (R1 != R2) then PC ← Address
10010	Jump Register	PC ← R1[7:0]
10011	Jump	PC ← Address
10100	Load Word	R1 = Mem[Address]
10101	Add Memory	R1 = R2 + Mem[Address]

10110	Store Word	Mem[Address] = R1
11010	Load Indirect	R1 = Mem[R2]
11011	Store Indirect	Mem[R2] = R3

Modules

Transparent register file

This module has the ability to include the write data in the its read result. If (Reg_Write=1 & Write_Reg_Num=Read_Reg_Num) then Read_Reg_Data=Write_Data; Otherwise act as normal RegisterFile

IFtoIDRegs, IDtoMemRegs, MemtoEXRegs, PostEXRegs:

These registers latch the registers and control signals between stages. The inputs of each of these registers is listed in the table below:

IFtoIDRegs:

- Inputs:
    - Instruction[31..0]
- Outputs:
    - Opcode[4..0]
    -

IDtoMemRegs

- Inputs:
    - Memory Stage Control Signals
    - Execution Stage Control Signals
    - Writeback Stage Control Signals
    - Register1 and Register2 and their indexes

    - Destination Register Index of Instruction
    - Shamt, Immediate Data and Address from Instruction
- Outputs:
    - Same as inputs

Preproccesser

This unit is responsible for inserting the micro-instructions in the pipeline. Whenever a instruction is going to be decomposed by this unit, the instruction that comes after this instruction must be stopped to make room in the pipeline for the decomposed instruction(It is very similar to a NOP insertion, but a micro instruction is inserted instead of a NOP). This module has a Wait signal as an output that does this. When an instruction that needs decomposing arrives, this

Hazard Detection Unit

This unit is in charge of detecting any hazards in the instructions that can not be avoided using the available forwardings.
Instructions that have potential unavoidable hazards include:

- Jump Register
- Store Word
- Store Word Indirect
- Load Word Indirect
- Single Add Indirect

*The last two instruction are from the extended ISA.

Forwarding Unit

This unit is used to determine whether a register needs forwarding or not. This unit receives up to 3 sources (not including the original source) along

with some flags that determine their validity. If one the forwarded sources is able to be forwarded, its means that the original source is