Ethan Wong
11/25/22

CS M151B HW4

866 10. 32 registers → 64 registers

lw, sw ↓12%. register file latency 15ps → 16ps cost 200 → 400

a) Calculate Clock cycle time before improvement

$C_1 = 250 + 150 + 25 + 200 + 150 + 5 + 30 + 20 + 50 + 50 = 930ps$

After improvement

$C_2 = 250 + 160 + 25 + 200 + 150 + 5 + 30 + 20 + 50 + 50 = 940ps$

Assuming 100 instructions: LDUR/STUR are 35%, decrease to 31%.

$\frac{930}{100} = 9.3$ $\frac{940}{96} = 9.79$ $\frac{9.3}{9.79} = 0.95$ Speedup is 0.95

b) Cost before improvement

$C_1 = 1000 + 200 + 10 + 100 + 30 + 2000 + 5 + 100 + 1 + 500 = 3946$

↳ cost per clock cycle before improvement

$\frac{3946}{930} = 4.24$

Cost after improvement

$C_2 = 1000 + 400 + 10 + 100 + 30 + 2000 + 5 + 100 + 1 + 500 = 4146$

↳ cost per clock cycle after improvement

$\frac{4146}{940} = 4.38$

The change in cost after improvements is 0.14.

Performance ratio = $\frac{4.38}{4.24} = 1.03$.

c) Adding more registers seems to increase the overall cost, but improves performance slightly. It would be beneficial to increase the number of registers if you are running a very large scale operation where performance must be optimized and there are ample computing resources to do so. It wouldn't make sense to add more registers if you only had a limited computer and performance didn't matter too much - i.e, for small tasks where a slight performance boost won't be noticeable.

11. Load with increment (r-type instruction) "lwi"

Reg[rd] = Mem[Reg[rs1] + Reg[rs2]]

- Add rs1 and rs2
- Go into memory and get the value stored at that number
- Store it into Reg[rd]

a) We don't need new functional blocks here.

b) Add an option to the control unit for the new "load with increment" instruction.

c) We don't need new data paths, the existing ones are sufficient.

d) We don't need new signals to execute this instruction.

lwi (R-format)

Opcode : 000000

RegDst:	1	going into Reg[rd]
ALUSrc:	0	no sign extension needed
MemtoReg:	1	Mem to Reg[rd]
RegWrite:	1	write to Reg[rd]
MemRead:	1	have to read Mem
MemWrite:	0	only writing to reg[rd]
Branch:	0	no branch
ALUOp1:	1	R-type instruction
ALUOp2:	0	R-type instruction

Path : Instruction [31-0]

Read Register 1 / Read Register 2

Read data 1 / Read data 2

Both Read data 1, Read data 2 go into ALU

ALU Result → Address, Read Data

Write data in Register

Done

12. swap rs, rt (I-type)

Reg[rt] = Reg[rs] ; Reg[rs] = Reg[rt]

a) No new functional blocks are needed for this instruction,

b) Modify the register file to allow swapping of the contents from two registers in one clock cycle.
ALU should support swapped read/write instructions,

c) We need a new data path that allows for the swap (from ALU).

d) We could use a "swap" signal from the control logic.

e) I would add a swap signal to the Control Logic that connects to the DATA A and DATA B outputs. When the swap signal is 1, it indicates we want to perform the swap. When swap is 1, the wires instead both branch to the WDATA in Register File, and they get swapped with each other. Reg[rt] = Reg[rs] first, then Reg[rs] = Reg[rt]. In order to do this we have to save both addresses as RegDst.

swap rs, rt

Opcode:	Something Unique	
RegDst:	0 :	We do write to Reg[rt] ...
ALUSrc:	0	No sign extension
MemtoReg:	0	No Memory
RegWrite:	1	Write to Reg[rt], Reg[rs]
MemRead:	0	No Memory
MemWrite:	0	No Memory
Branch:	0	No Branch required
$ALUOp_1$:	X	ALU is avoided
$ALUOp_2$:	X	ALU is avoided
Swap:	1	

MEM[ReadData1] = Read data 1 + Instruction[15-0]

13. ss rt, rs, imm (Store Sum) → I-type

Mem[Reg[rt]] = Reg[rs] + SE(immediate)

- Add Reg[rs] + immediate
- Store in Memory at [Reg[rt]] address

a) Yes, we need new multiplexers to modify the data path so we can store the sum in registers.

b) The ALU needs to be modified so it can store the sum and support read/write.

c) A new datapath is needed to implement the changes to the ALU.

d) We need a new signal called "ss" that tells the datapath when we are performing store sum

e) I would add a "ss" to the control logic so the datapath knows when to store sum. It would connect to the ALU and DM. When "ss" is 1, the ALU can take three inputs. It will take Data B (Reg[rt]) through a new multiplexer, only when "ss" is 1. Data A and the SE immediate will be added as usual in the ALU. Next, the address from the ALU will go into the ADDR portion of the DM. The "ss" signal into the DM will tell the DM to use that address for writing, and the sum will be written to that address in memory.

ss rt, rs, imm

Opcode: Something Unique

RegDst:	X	No register destination, writing to Memory
ALUSrc:	1	need sign extension
MemtoReg:	0	not writing to Reg
RegWrite:	0	not writing to Reg
MemRead:	0	not reading from memory
MemWrite:	1	write at Mem[Reg[rt]]
Branch:	0	No branch
ALUOp1:	1	} need addition
ALUOp2:	0	